Analysis of Security Protocols by Annotations

Gao, Han

Publication date:
2008

Document Version
Publisher's PDF, also known as Version of record

Analysis of Security Protocols by Annotations

Han Gao

Kongens Lyngby 2008
IMM-PHD-2008-190

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Abstract

The trend in Information Technology is that distributed systems and networks are becoming increasingly important, as most of the services and opportunities that characterise the modern society are based on these technologies. Communication among agents over networks has therefore acquired a great deal of research interest. In order to provide effective and reliable means of communication, more and more communication protocols are invented, and for most of them, *security* is a significant goal.

It has long been a challenge to determine conclusively whether a given protocol is secure or not. The development of formal techniques, e.g. control flow analyses, that can check various security properties, is an important tool to meet this challenge. This dissertation contributes to the development of such techniques.

In this dissertation, security protocols are modelled in the process calculus LySa. A variety of interesting security properties that protocols are often expected to have are formalised: authentication, confidentiality, freshness, absence of simple and complex type flaws. Those security properties are explicitly specified as *annotations* embedded in the LySa syntax. Finally, a number of automatic techniques for the analysis of system behaviour are developed. These techniques are specified as control flow analyses and are, therefore, guaranteed to terminate.

The perspectives for the analysis techniques are discussed. Thus the dissertation marks a step forward both for scientists, who gain a general framework for the study of several interesting security properties, and developers, who get a collection of tools that can validate protocols with respect to various aspects of security.

Resumé

Tidens informationsteknologiske trend er at distribuerede systemer og netværk bliver mere og mere vigtige, idet de fleste af de services og muligheder, der tegner det moderne samfund, er baseret på disse teknologier. Netværksbaseret kommunikation mellem autonome agenter har derfor fået stor forskningsmæssig interesse. For at tilvejebringe effektive, pålidelige og sikre kommunikationsformer bliver der til stadighed udviklet et væld af protokoller. I de fleste af disse protokoller er sikkerhed et væsentligt mål.

Det har længe været en udfordring at give garantier for om en given protokol er sikker. Udviklingen af formelle analyser, der kan verificere forskellige ønskværdige sikkerhedsegenskaber, er derfor et vigtigt redskab når denne udfordring skal imødegås. Denne afhandling bidrager til udviklingen af sådanne teknikker.

I afhandlingen modelleres sikkerhedsprotokoller i proces kalkulen LySa. Dernæst formaliseres et antal vigtige sikkerhedsegenskaber, som protokoller normalt forventes at have: autenticitet (authentication), fortrolighed (confidentiality), friskhed (freshness) og fraværet af både simple og komplekse typefejl (type flaws). Disse egenskaber specificeres eksplicit i form af annoteringer der indføjes i protokollernes specifikationer ved brug af udvidelser af LySa kalkulens syntaks. Det sidste skridt består i udviklingen af automatiske teknikker til analyse af systemers opførsel. Disse teknikker er specificeret som statiske kontrol flow analyser og er derfor fuldt automatiske og terminerer altid.

De brugsmæssige såvel som videnskabelige perspektiver for disse teknikker behandles. Dermed bidrager afhandlingen med et skridt fremad både for forskere, som får en mere generel ramme, indenfor hvilken interessante sikkerhedsegen-

skaber kan studeres, og for udviklere, som får en samling af værktøjer, der kan validere kryptografiske protokoller i forhold til forskellige sikkerhedsaspekter.

Preface

This thesis was prepared at Informatics Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring the Ph.D. degree in engineering. The Ph.D. study has been carried out under supervision of Professor Hanne Riis Nielson in the period from September 2004 to January 2008.

Acknowledgement I would like to thank Hanne Riis Nielson and Flemming Nielson for the supervision and for providing me with this opportunity to work on an international research project.

Furthermore I thank the rest of the LBT group: Terkel Tolstrup, Henrik Pilegaard, Jörg Bauer, Christoffer Rosenkilde Nielson, Sebanstian Nanz for good company. A special thank goes to Jörg Bauer for proof reading some chapters of this thesis and to Henrik Pilegaard and Sebanstian Nanz for commenting and translating the Abstract.

I thank to Professor Pierpaolo Degano and Chiara Bodei for taking good care of me during my visit to Pisa.

Finally, I owe a lot to my family, especially my husband Lei Zhang, for the patient and support. I could not have finished this thesis without it.

Lyngby, January 2008

Han Gao

Contents

Chapter 1

Introduction

In modern times, distributed systems and networks are becoming increasingly important, upon which most technologies that shape today's society are built. Communication among agents over networks is then drawing a lot of attention. In order to build efficient and reliable means of communication, more and more protocols have been invented. In most of the protocols, *security* plays a significant role and is normally a goal to achieve. For example, when shopping online the credit card information of customers has to be kept secret, and when doing e-banking, a money transfer request has to be recognized as coming from a person who has the right to do it. In these two examples, security involves confidentiality and authenticity, respectively. However there are more aspects of security, e.g. integrity, availability and non-repudiation, depending on the application in question.

Generally speaking, in distributed systems and networks, security problems are undecidable [30] for their dynamic behaviour due to, say, mis-behaved agents and unbounded sessions of protocol executions [35, 42, 26, 76]. Therefore, verification of security properties is an important research problem. This leads to the researches in searching for a way to verify whether a system is secure or not.

Formal methods offer a promising approach for automated security analysis of protocols: the intuitive notions are translated into formal specifications, which is essential for a careful design and analysis, and protocol executions can be

simulated, making it easier to verify certain security properties.

This thesis relies on the use of formal methods and investigates one way to soundly validate protocols. The aim of this thesis is to illustrate that:

> The intended behaviour of cryptographic protocols can be made more explicit using annotations. Such annotations will facilitate static program analysis to validate a number of important security properties the protocols are supposed to adhere to.

1.1 Background

The central topic of this thesis is composed of the following elements.

1.1.1 Security Protocols

The most frequently used way of protecting communication is by means of cryptography, i.e. data from one party is encrypted using a key before sending it out onto the network and the other party who receives the encryption has to use a corresponding key to decrypt the message before being able to read the data. Obviously it requires that these two parties have to agree on the keys used to encrypt and decrypt prior to performing any action. The sequence of message exchanges in order to generate and distribute cryptographic keys to the intended users is the so called *security protocol* (or *communication protocol, cryptographic protocol*).

Designing security protocols is complex and often error prone: various attacks are reported in the literature to protocols thought to be "correct" for many years. This is due to the nature of protocols: they are executed in an uncertain environment, where some of the participants are not fully trusted or even maybe malicious. Sometimes, the attackers do not need to break cryptography. Indeed, these are the cases considered in this thesis, i.e. the underlying cryptography of protocols is assumed to be *perfect*, which means that security issues regarding cryptographic primitives and specific cryptographic algorithms are abstracted away, and protocols are broken merely because of attackers exploiting flaws in the protocols. The attacker is powerful enough to perform a number of potentially dangerous actions: he is able to intercept messages flowing over the network, or replace them by new ones using the knowledge he has previously gained; he is able to perform encryption and decryption using the keys within

his knowledge. Consequently, albeit carefully designed, security protocols may have various flaws , allowing the attackers to break it.

Each security protocol is designed to achieve certain goals after the execution. Those goals are called *security properties*. There are various security properties, for example, to ensure that secret data is not revealed to irrelevant parties, to make two parities share a same fresh key, or to make sure they are the intended senders/receivers. Due to the presence of an attacker, protocols are easily subject to some flaws and thus not able to preserve the expected security properties. Therefore it is very important to find a formal way to prove their correctness with respect to security properties. Indeed, this is the essential goal of this thesis.

Security protocols are widely used. Example protocol applications include the following,

e-shopping Shopping on the Internet becomes more and more popular now because of its convenience. It offers lots of benefits that cannot be found when shopping in a store or by mail: the Internet is always open – seven days a week, 24 hours a day and bargains can be numerous. Buying an airline ticket, booking a hotel, sending flowers to a friend can be done within a few seconds: just select what you want and send out your credit card number. However, one crucial concern of e-shopping is the secrecy of the credit card number such that it won't be known and abused by a third party. To this aim, most of the online shops adopt the Secure Sockets Layer (SSL V3.0) protocol, a security protocol that provides communication privacy over the Internet. The protocol is used to establish a secure connection between two parities and hence allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

e-voting Voting is one of the foundational aspects of any democratic society. It is much more complicated than simply counting votes. In fact, most of the work in an election occurs long before/after the voter ever steps into the booth, e.g. ballot preparation is a long process that is complicated by millions of rules and regulations, and a huge number of volunteer workforce is involved. For the United States of America, the total country election expenditures, in 2002, were estimated at over 1 billion dollars, or about 10 dollars per voter. Due to the large amount of votes, the traditional, i.e. paper-based, voting system is very vulnerable and may have some disadvantages compared to electronic voting systems, e.g. Charles Stewart of the Massachusetts Institute of Technology estimates that 1 million more ballots were counted in 2004 than in 2000 because electronic voting machines detected votes that paper-based machines would have missed.

Naturally, an electronic voting system is embedded with voting protocols, which try to use advanced cryptography to make electronic voting secure and meanwhile guarantee e-voting requirements, e.g. democracy (the voters can vote only once and only the voters can vote), accuracy (a voter's vote cannot be altered, duplicate, or removed without being detected) and privacy (votes remain anonymous).

1.1.2 Process Calculi

Process calculi (or process algebras) are programming-like languages for describing concurrent and distributed systems. There are a number of different, but somehow related, process calculi, e.g. CSP [50], CCS [67] as well as some recent members of this family, $\pi-$calculus [66], spi-calculus [3] and the ambient calculus [25].

While the variety of existing process calculi is very large, e.g. some incorporate stochastic behaviour and some include timing information, they do share several common features: they consist of active components, also known as processes; each process can perform its own actions independently or interact with other processes by communication (message passing); processes can be combined by some basic operators, e.g. sequentialisation, parallel compositions and sometimes non-deterministic choice; and processes can declare scopes of terms using restriction operators.

Parallel composition is a key primitive to distinguish process calculi from sequential computation. Assuming we have two processes P and Q, parallel composition of these two processes is normally represented by $P \mid Q$, which allows computation in P and Q to proceed simultaneously and independently, and also allows interaction between P and Q.

Interaction (or communication) can be viewed as a directed flow of information from an outputting process (e.g. $\langle t \rangle$) to an inputting processes (e.g. $(; x)$). The outputting process has to specify the data to be sent, which is t in $\langle t \rangle$. Similarly, if an input expects to receive data, one or more bound variables will act as place-holders to be substituted by data, when it arrives. In $(; x)$, x plays that role.

Sequential composition can be used to regulate the orders in which interactions are executed. For example, it might be desirable to specify that: first to receive data on x and then send that data out. In process calculi, the sequentialisation operator, normally represented by ., is usually integrated with input or output, or both. For example a process $(; x).P$ will wait for an input, only when this

input has been received will the process P be activated, with the received data substituted for x.

One advantage of using process calculi to model concurrent systems is that they are always equipped with formal semantics specifying how the system evolves. As far as process calculi are concerned, they are often given a structural operational semantics (also called small step semantics). The semantics is regarded as "syntax-directed" such that each compositional construct is defined in terms of individual components, following various syntactic possibilities. The behaviour and evolution of a process are described by means of a transition system and each computational step is defined as a path in the transition system. Transitions are defined using axioms and inference rules of the form,

$$\frac{Premise_1 \wedge \ldots \wedge Premise_k}{Conclusion}$$

which can be read as "the conclusion holds when all the k premises are satisfied".

Example 1.1 Assume there is a simple semantics rule:

$$\langle t \rangle.P \mid (;x).Q \rightarrow P \mid Q[t/x]$$

The interpretation of this rule is:

1. The process $\langle t \rangle.P$ sends out a message, t. Dually, the process $(;x).Q$ receives that message. Once the message has been sent, $\langle t \rangle.P$ becomes the process P, while $(;x).Q$ becomes the process $Q[t/x]$, which is Q with the place-holder x substituted by the received data t.

Note that there is no premise in this case.

Process calculi can be used to model various concurrent systems, e.g. biological systems, mobile ambients and security protocols. Executions of cryptographic protocols can be described as a sequence of internal operations performed by each principal and synchronised by their communications. Naturally this can be modelled in a process calculus at a very abstract level. The desired security properties are then studied by checking their specification along all the possible computational steps. A large amount of research has been done in recent years using various process calculi, e.g. to establish properties about the information flow and to detect flaws in cryptographic protocols. It is also the approach adopted in this thesis, i.e. using a process calculus to formally model security protocols and specify security properties.

1.1.3 Program Analysis

A program is any formal notation used to communicate ideas precisely between people or between a person and a machine. Program analysis provides a static way of deriving behavioural information of programs arising dynamically at run-time without actually executing it. The goal of program analysis is thus to produce descriptions of certain program properties applicable to all possible executions of the program for arbitrary input values. Example program properties include: (a) which program points are reachable by variable x, and (b) which are the possible values of variable y at run time. However, according to Rice's Theorem [75], some properties of programs are undecidable, including the above examples. In order to keep analysis results computable, program analysis can only provide *safe approximate* answers and therefore efforts have to be made to keep the approximation as close to the precise result as possible. However, in general, program analysis is very efficient in the sense that it is focusing on the fully automatic processing of large programs and the computations are guaranteed to terminate by lattice and fix-point theory [82].

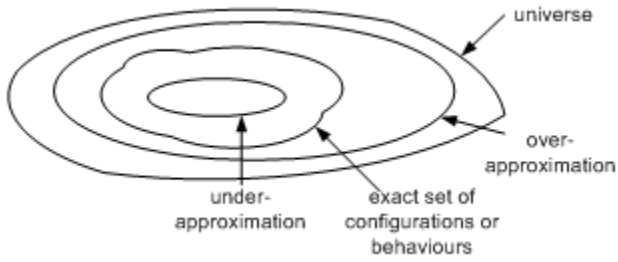


Figure 1.1: Nature of Program Analysis

There are two kinds of approximations,

- under-approximation, which estimates the program behaviour, that *must* happen along all the execution paths; and
- over-approximation, which estimates the program behaviour that *may* happen along at least one execution path.

Choosing the right approximation depends on which property of a program needs to be checked.

Example 1.2 The program fragment,

$$
\begin{array}{l}
\text{read}(x); \\
\text{if } x > 0 \\
\quad \text{then} \quad y := 2 \\
\quad \text{else} \quad y := 0; \\
z := 5/y
\end{array}
$$

will firstly read a user input x, then check whether x is a positive number or not before assigning different values to y and finally compute the value of z. Assume one wants to ensure that all the *divide by zero* errors are found before actually executing the program. Obviously it is of interest to estimate which value y holds after the last statement. An over-approximation of the value of y may be $\{0, 2\}$, which is the most precise result for this program, or $\{0, 2, 45\}$, which is a safe but less precise one. On the other hand, $\{2\}$, for example, is not a safe over-approximation, because it does not contain all the values that y may possibly hold. Furthermore, $\emptyset$ is a safe under-approximation of y's value, because none of the values is guaranteed to be held by y after the last statement.

The results of program analysis can be used to, for example, optimise compilers for generating small, faster, less memory consuming code, or debug programs for finding dead code or type-mismatched function calls. A more recent application of program analysis is to capture malicious or unintended behaviours of software and systems. In this thesis, we shall use program analysis techniques to validate security protocols with respect to different security properties.

There are several approaches to program analysis, including *Data Flow Analysis*, *Control Flow Analysis*, *Abstract Interpretation* and *Type and Effect Systems*. These approaches are divided into different classes depending on whether they compute the flow of data or the flow of control in a program.

However, there is a large amount of commonality among these approaches. A comparison between them may help with choosing the right one for checking the property in question and exploiting insights gained from one approach may enhance the power of other approaches.

In this thesis, we restrict ourselves to Control Flow Analysis, a specific static technique, based on Flow Logic for studying various security properties of communication protocols.

1.1.4 Flow Logic for Control Flow Analysis

Flow Logic [74] is a relatively new program analysis technique. It is actually based on existing technologies and insights, especially from the classical areas of data flow analysis, constraint based analysis and abstract interpretation. Flow logic is concerned about specifying which criteria an analysis estimate should satisfy in order to be acceptable for a program.

A very important feature of flow logic is that it separates the specification of an acceptable estimate from the actual computation of the analysis information. This abstraction enables it to be applied to a variety of programming paradigms, including imperative, functional, object-oriented and concurrent constructs, and even a combination of different paradigms. From flow logic's point of view, an analysis is split into two phases:

1. define a judgement, usually in clausal form, expressing the acceptability of an analysis estimate for a program.
2. compute the clauses to get a solution.

One advantage of such a division is that the second phase is independent of the kind of analyses as well as programming languages, therefore many state-of-the-art technologies may be employed for computing the analysis information, including the analysis of mobile ambients [25], Java cards [46] and security protocols [15]. Furthermore, it also makes specification and implementation become independent. Thus one can concentrate on specifying the analysis without considering the implementation.

In this thesis, we will continue to exploit flow logic by applying it to specify control flow analysis for communication protocols.

A control flow analysis of a program P works by collecting information about some aspects of the behaviour of the program P. This behavioural information is stored in a data structure, also called *analysis component*, say $\mathcal{A}$. An analysis in flow logic style is syntax-directed: it is defined structurally on the syntax of the program P by giving rules of the form

$$\mathcal{A} \models P \quad \text{iff} \;\; a\ logic\ formula\ holds$$

meaning that $\mathcal{A}$ is a description of the behaviour of the program P when the logic formula holds. The logic formula can be any arbitrary one and may even be recursive, e.g. containing $\mathcal{A} \models P'$ for an arbitrary program P'.

Example 1.4 Assume one wants to analyse which data may be sent over the network by a process. The analysis in flow logic style can be defined as:

$$\kappa \models \langle t \rangle.P \;\; \text{iff} \; \langle t \rangle \in \kappa \;\wedge\; \kappa \models P$$
$$\vdots$$

where κ is an analysis component containing all the messages sent over the network. It can be read as "in order for κ to be a valid analysis estimations of the process $\langle t \rangle.P$, it has to be the case that 1) $\langle t \rangle$ is contained in κ, and 2) κ is also a valid estimation of the continuation process P".

1.2 Contribution Outline

In this context this dissertation contributes a number of static analyses in support of the main thesis. Each of the static analyses focuses on one of the security properties of communication protocols, which are confidentiality, freshness, simple type flaw free, and complex type flaw free. There are also a number of publications present these works; they are confidentiality [38], freshness [37, 36], simple type flaw free [17, 14], and complex type flaw free [79]. These analyses constitute a nice toolbox for the analysis of communication protocols, which may provide useful information about security properties.

1.3 Overview of the Thesis

Chapter 2 gives an introduction to some basic concepts related to cryptographic protocols as well as a list of protocol narrations, which are analysed in this thesis.

Chapter 3 introduces the process calculus LySA, which is a calculus that models systems using secure network communication protected by means of cryptography. Both the syntax and semantics are presented. It also shows how to encode in LySA protocols executed in different scenarios.

Chapter 4 introduces the basic concepts of the analysis technique and shows how analyses are formulated in the Flow Logic framework. Next, a fairly standard control flow analysis of LySA is given that captures the entire behaviour of any LySA process. It is illustrated how to prove within the Flow Logic framework that the analysis is indeed able to capture the behaviour of a process with respect to the formal semantics given in Chapter 3.

Chapter 5 concerns the authentication property of protocols. Both the LySA calculus, presented in Chapter 3, and the standard Control Flow Analysis, presented in Chapter 4, are extended to include orig and dest annotations in order to facilitate capturing authentication violations.

Chapter 6 analyses the confidentiality properties and extends standard LySA and it's Control Flow Analysis with within and from annotations to detect any confidentiality violations.

This work was previously presented in [38].

Chapter 7 describes how to detect replay attacks. It is done by distinguishing terms from different sessions. To this aim, the standard LySA calculus is decorated with session identifiers and the Control Flow Analysis is extended to capture message replays by inspecting the session identifiers.

This work was previously presented in [37, 36].

Chapter 8 deals with the simple type flaw attacks, which happen when a field in a message is interpreted as of an unexpected type. The standard LySA calculus is extended to include types, terms are associated with their types. The Control Flow Analysis is then extended to compare whether the received term is of the expected type and hence captures any type mismatching.

This work was previously presented in [17, 14].

Chapter 9 discusses complex type flaw attacks, the central idea of which is to fool a principal by misinterpreting the fields and accepting a concatenation of fields as single field. The notion of one-to-one variable binding in the standard LySA calculus is relaxed in order to capture multi-to-one bindings. It is guaranteed that a protocol is free of complex type flaw attacks if there are no possible multi-to-one bindings.

This work was previously presented in [79].

Chapter 10 concludes the thesis and discusses perspectives of future work.

Chapter 2

Cryptographic Protocols

Cryptographic protocol, as suggested by the name, employs cryptographic algorithms to protect sensitive data. It normally consists of a sequence of message exchanging and is to achieve some security-related goals after a complete execution.

Before going into the details, we shall describe in words a very simple cryptographic protocol as an example.

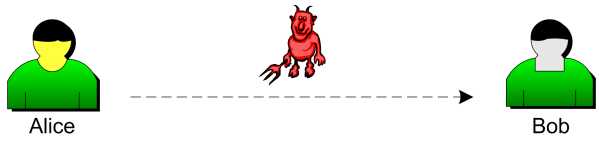


Figure 2.1: A Simple Protocol

Example 2.1 *In our scenario, as shown in Figure 2.1, there are two users, say Alice and Bob, where Alice wants to send to Bob a secret data, credit card information for instance, over an insecure network. Assuming both Alice and Bob pre-share a key in advance, one way to achieve the goal is that Alice uses the key to encrypt the secret data and sends the encryption to Bob along the network. When receiving the encryption, Bob then decrypts it using the key and*

gets hold of the secret data. After successfully completed the protocol, Alice and Bob, and nobody else, should know the secret data.

In the above example, cryptography is used to protect the security of data. The sender encrypts the sensitive data using the key and the receiver at the other end of the network decrypts the message using the corresponding key to extract the data. However, as pinpointed by [41],

> cryptography is rarely ever the solution to a security problem. Cryptography is a translation mechanism, usually converting a communication security problem into a key management problem and ultimately into a computer security problem. Hopefully, the resulting problem is easier to solve than the original problem. In summary, cryptography can enhance computer security; it is not a substitute for computer security.

The reason is that there are various kinds of attacks, which can prevent cryptographic protocols from achieving the expected security goals without breaking the underlying cryptographic algorithms.

2.1 Security Attacks

What kind of attacks do there exist against security protocols? This question cannot be answered before having defined what we expect from a given security protocol. As an example, assume that the purpose of a certain protocol is to distribute credit card information safely between, say, Alice and Bob (as in Example 2.1). Then we expect that after the protocol has been successfully terminated, the credit card information is known to Alice and Bob, only. If the protocol can terminate successfully so that an outsider knows the information then our expectations have not been realized. There is an attack against the protocol.

Let us now enumerate some typical attacks. In general, the communication over network is viewed as a flow of information from a source, such as a file or a user, to a destination, such as another file or a user. The attacks on the security of network can be categorised into the following [55].

Interruption The communication flow is destroyed or becomes unavailable or unusable. Examples include destruction of a piece of hardware, i.e. a hard

disk, or the cutting of a physical communication line, i.e. a cable.

Eavesdropping An unauthorised party gains access to the communication. The unauthorised party could be a person, a program, or a computer. Examples include wiretapping to capture data in a network, and the illegally copying of files or programs.

Fabrication An unauthorised party inserts counterfeit data into the communication flow. Examples include the inserting of spurious message in a network or the addition of records to a file.

Modification An unauthorised party not only gains access to but tampers with the communication flow. Examples include changing values in a data file, altering a program so that it performs differently, and modifying the content of messages being transmitted in a network.

Traffic analysis An unauthorised party intercepts and examines the messages flowing over the network in order to deduce information from the message patterns. It can be performed even when the messages are encrypted and can not be decrypted.

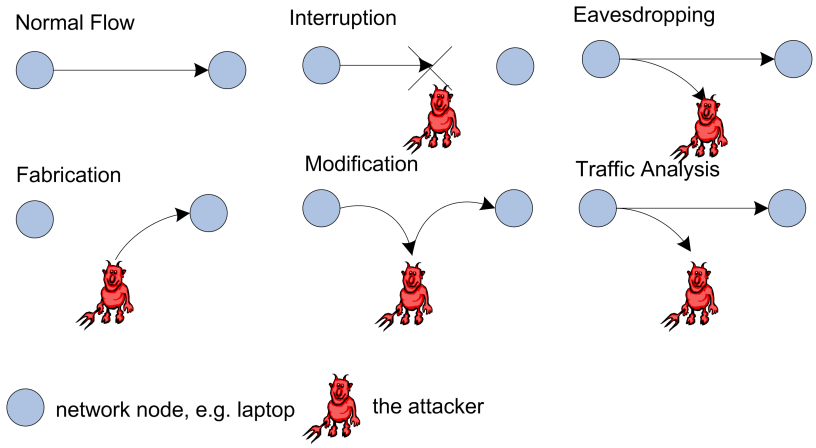


Figure 2.2: Graphical Representation of the attacks

These attacks can be represented as in Figure 2.2.

Attacks can be divided into active and passive attacks. Passive attacks are attacks that can be carried out by eavesdropping on the communications between the participants, without modifying or redirecting them. All other attacks are active attacks. Typically, passive attacks are somehow *trivial* and almost all

attacks that are mentioned in the literature are the active ones, which is also the case in this thesis.

A protocol execution is considered as involving honest principals and active attackers. The abilities of the attackers and relationship between honest principals and attackers together constitute a *threat model*. The almost exclusively used threat model is the one proposed by Dolev and Yao [33], which is therefore often referred to as *Dolev-Yao threat model*. The Dolev-Yao threat model is a worst-case model in the sense that the network, over which the principals communicate, is thought as being totally controlled by an omnipotent attacker with all the capabilities listed above. Therefore, there is no need to assume the existence of multiple attackers, because they together do not have more abilities than the single omnipotent one. Dishonest principals do not need to be considered either: they can be viewed as attackers. Furthermore, obviously, it is not interesting to consider an attacker with less abilities than the omnipotent one.

Example 2.2 *In Example 2.1, the attacker can eavesdrop on the encryption sent over the network. In case he is in possession of the pre-shared key between Alice and Bob, he is able to attack the protocol in, at least, the following two ways:*

1. *eavesdrop the communication and perform decryption and therefore knows the credit card information.*
2. *intercept the communication and encrypt something else and send to Bob, who will then be cheated in accepting a faked credit card information.*

2.2 Security Properties

Each cryptographic protocol is designed to achieve one or more security-related goals after a successful execution, in other words, the principals involved may reason about certain properties; for example, only certain principals have access to particular secret information. They may then use this information to verify claims about subsequent communication, e.g. an encrypted message can only be decrypted by the principals who have access to the corresponding encryption key.

The most commonly considered security properties include,

Authentication Authentication is concerned with assuring that a communication is authentic. In the case of an ongoing interaction, such as the

connection of a host to another host, two aspects are involved. First, at the time of connection initiation, the two entities have to be authentic, i.e. each is the entity that he claims to be. Second, during the connection, there is no third party who interferes in such a way that he can masquerade as one of the two legitimate parties for the purposes of unauthorized transmission or reception. For example, fabrication is an attack on authenticity.

Confidentiality Confidentiality is the protection of transmitted data from attacks. With respect to the release of message contents, several levels of protection can be identified, including the protection of a single message or even specific fields within a message. For example, interception is an attack on confidentiality.

Integrity Integrity assures that messages are received as sent, with no duplication, insertion, modification, reordering, or replays. As with confidentiality, integrity can apply to a stream of messages, a single message, or selected fields within a message. Modification is an attack on integrity.

Availability Availability assures that a service or a piece of information is accessible to legitimate users or receivers upon request. There are two common ways to specify availability. The first approach is to quantify system reliability using measurable criteria, such as the failure probability or the MTTF/MTTR (mean time to fail/ mean time to recover) ratio [78]. The second approach is to specify failure factors (factors that could cause the system or the communication to fail) [77], for example, the minimum number of host failures needed to bring down the system or the communication. Interruption is, for example, an attack on availability.

Non-repudiation Non-repudiation prevents either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the message was in fact sent by the alleged sender. Similarly, when a message is received, the sender can prove that the message was in fact received by the alleged receiver.

Example 2.3 *Consider the two attacks on the protocol in Example 2.2. The first attack breaks the confidentiality of the credit card information, because it is known to the attacker. The second attack breaks the authentication of Alice to Bob: after completing the protocol, Bob believes that what he received is from Alice but actually it is from the attacker instead.*

2.3 Security Mechanisms

Cryptographic mechanisms are fundamental to cryptographic protocols. Suppose that we have a message text M that we wish to transmit over the network. The process of converting M to a form that is not understandable to anyone monitoring the network is called *encryption*. This conversion depends on an additional parameter K known as the *key*.

The intended receiver of an encrypted message may wish to recover the original text M. To do this, a second key (K or K^{-1}) is used to reverse the process. This reverse process is known as *decryption*.

It is very obvious that by restricting appropriately who has access to the keys involved, we can limit the ability to encrypt or decrypt messages.

2.3.1 Cryptographic Primitives

Symmetric Key Cryptography In symmetric key cryptography, the encryption key and decryption key are identical. Of course, anyone who holds the key can create encryption and read the contents of encrypted messages. To ensure security of communication this key is kept secret between the communicating principals.

In this thesis, we shall represent the encryption of M using key K by $\{M\}_K$ and adopt the notation of perfect encryption, where we do not consider the attacker to be all powerful in terms of solving computational problems. This means that there are certain things that the attacker can not do, e.g. without the correct key, the attacker cannot decrypt an encryption or generate an encryption without using the proper key.

Asymmetric Key Cryptography In asymmetric key cryptography (or public key cryptography) there is no shared secret between communicating principals. In this scenario, each principal is associated with a key pair (K, K^{-1}). The public key K is made publicly available but the private K^{-1} is kept secret from anybody else. Any principal can encrypt a message M using K and only the key-holder can then decrypt it using K^{-1}. Thus, the secrecy of messages to the principal can be ensured.

Some public key algorithms allow the private key to be used to encrypt messages with the public key being used to decrypt the corresponding encryption. If an

encryption decrypts (using K) to a meaningful message M then it is assumed that the encryption must have been created by the key-holder using the key K^{-1}. This can be used to guarantee the authenticity of the message. Such algorithms are often said to provide a digital signature capability.

One-way Hash Function We shall often require evidence that a message that has been sent has not been subject to modification in any way. Typically this is carried out by using a hash function. A hash function H when applied to a message M yields a value $H(M)$ known as the hash value of that message. The mapping of message to its hash result is one-way; given M and $H(M)$ it should be computationally infeasible to find M' such that $H(M') = H(M)$. Therefore a receiver of a message can check whether a message and a corresponding hash result agree. Hash functions are largely intended for use in conjunction with cryptography to provide signatures.

2.3.2 Session Key vs. Master Key

In general, the cryptographic protocol architecture consists of two principals, who are willing to engage in a secure communication using an insecure network, and a trusted server, which generates the session key used for exchanging data securely between the principals. The session key is abandoned after data exchanging is over. In fact, it is not possible to establish an authenticated session key without existing secure channels already being available [19]. Therefore it is essential that some keys are already shared between different principals, which are often referred to as master keys. Different from session keys, which expire after each session, master keys are changed less frequently, and consequently leaking master keys always causes cryptographic protocols to be broken.

2.4 Protocol Narrations

Protocols are normally expressed as narrations, where some finer details are abstracted away. A protocol narration is a simple sequence of message exchanges between the different participating principals and can be interpreted as the intended trace of the ideal execution of the protocols. Consider the following example.

Example 2.4 *The protocol narration of Example 2.1 can be written as*

$$1. \quad A \rightarrow B: \quad \{CCI\}_K$$

where A is Alice for short, B is Bob and CCI stands for the credit card information. The protocol narration specifies that the protocol has only one step that A sends to B the credit card information encrypted using the key K.

2.4.1 Some Example Protocol Narrations

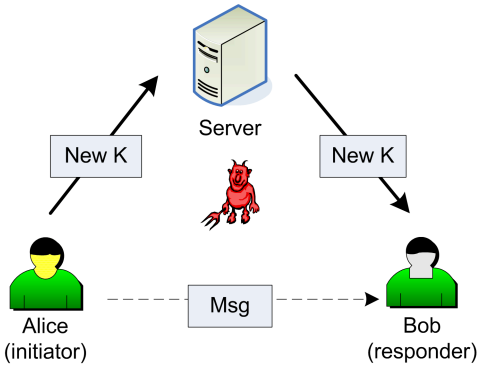
We collect in this section all the protocols being mentioned or analysed in the thesis.

2.4.1.1 Wide Mouthed Frog protocol

WMF is a symmetric key management protocol aiming at establishing a secret session key K_{ab} between the two principals A and B sharing secret master keys K_A and K_B, respectively, with a trusted server S. The protocol is specified by the informal narration below.

In the first message A sends to S its name, and then a fresh key K_{ab} and the name of the intended receiver B, encrypted under the key K_A. In the second one, S forwards the key and the sender name A to B, encrypted under the key K_B. Finally, B sends A the message Msg encrypted under the session key K_{ab} (Note that usually A sends the last message).

1. $A \rightarrow S: \quad \{K\}_{K_a}$
2. $S \rightarrow B: \quad \{K\}_{K_b}$
3. $B \rightarrow A: \quad \{Msg\}_K$



2.4.1.2 Andrew Secure RPC protocol

The goal of the Andrew Secure RPC protocol is to exchange a fresh, authenticated, secret key between two principals sharing a symmetric key K. It has

four steps. In the first message the initiator A sends a nonce N_A, the responder B increments and returns it as the second message together with his nonce N_B. A accepts the value and returns the $N_B + 1$, B receives and checks the third message and if it contains the nonce incremented, then he sends a new session key, K' to A together with a new value N'_B to be used in subsequent communications. The protocol narration is the following,

$$\begin{array}{lll} 1. & A \rightarrow B: & A, \{N_A\}_K \\ 2. & B \rightarrow A: & \{N_A + 1, N_B\}_K \\ 3. & A \rightarrow B: & \{N_B + 1\}_K \\ 4. & B \rightarrow A: & \{K', N'_B\}_K \end{array}$$

The Andrew Secure RPC protocol is subject to a simple type flaw attack as shown below: by replaying the message from step 2 to B in step 4, the attacker can successfully force A to accept $N_A + 1$ as the new session key. The protocol makes use of an operation to increment N_A, in step 2, and N_B, in the third step [1]. The attack is shown below,

$$\begin{array}{lll} 1. & A \rightarrow B: & A, \{N_A\}_K \\ 2. & B \rightarrow A: & \{N_A + 1, N_B\}_K \\ 3. & A \rightarrow M(B): & \{N_B + 1\}_K \\ 4. & M(B) \rightarrow A: & \{N_A + 1, N_B\}_K \end{array}$$

An improved version of Andrew Secure RPC protocol is suggested in [23], the so called BAN version, in order to prevent the above mentioned type flaw attack. The fixing amounts to inserting another component N_A into the encryption in the fourth message, as shown below,

$$4'. \quad B \rightarrow A: \quad \{K', N'_B, N_A\}_K$$

and leave the other steps untouched.

[1] The increment operation in LySa can be modelled as an encryption with key $SUCC$, where $SUCC$ is known to the attacker.

2.4.1.3 Needham-Schroeder Symmetric Key protocol

1. $A \rightarrow S: \quad A, B, N_a$
2. $S \rightarrow A: \quad \{N_a, B, K, \{K, A\}_{K_b}\}_{K_a}$
3. $A \rightarrow B: \quad \{K, A\}_{K_b}$
4. $B \rightarrow A: \quad \{N_b\}_K$
5. $A \rightarrow B: \quad \{N_b - 1\}_K$
6. $A \rightarrow B: \quad \{Msg\}_K$

This protocol involves two principals A and B and a server S, with whom A and B share the master keys k_a and k_b, respectively. The protocol has 6 steps to accomplish the goal that a session key K is generated by S and distributed to both A and B. The protocol makes use of nonce, typically a random number only used once and N_a and N_b in this example, to ensure the freshness of the messages. In step 1, A tells S that she wants to talk to B and uses the nonce N_a to ensure that this request is not a replayed one. Upon receiving the request, S, in step 2, generates a new session key K and sends it back to A, as encrypted together some other information using the master key K_a. In step 3, A decrypts the messages and forwards the field, $\{K, A\}_{K_b}$ to B. In step 4 and 5, B identifies A by challenging her using the nonce N_b. If A answers the challenge correctly, they then communicate with each other using K as the session key.

However, three years after invented, the Needham-Schroeder protocol was found to be flawed; it subjects to a replay attack in case an old session key, K', is revealed to the attacker.

1. $A \rightarrow S: \quad A, B, N_a$
2. $S \rightarrow A: \quad \{N_a, B, K, \{K, A\}_{K_b}\}_{K_a}$
3. $M(A) \rightarrow B: \quad \{K', A\}_{K_b}$
4. $B \rightarrow M(A): \quad \{N_b\}_{K'}$
5. $M(A) \rightarrow B: \quad \{N_b - 1\}_{K'}$
6. $M(A) \rightarrow B: \quad \{Msg\}_{K'}$

In the above example, $M(A)$ represents the attacker, who pretends to be the principal A and comes into play from message 3. By knowing the old session key K', he is able to replay the message $\{K', A\}_{K_b}$ from a previous session and therefore later on force B to accept K' as a fresh session key.

Amended Needham-Schroeder protocol:

1. $A \rightarrow B: \quad A$
2. $B \rightarrow A: \quad \{A, N_B\}_{K_B}$
3. $A \rightarrow S: \quad A, B, N_A, \{A, N_B\}_{K_B}$
4. $S \rightarrow A: \quad \{N_A, B, K, \{K, N_B, A\}_{K_B}\}_{K_A}$
5. $A \rightarrow B: \quad \{K, N_B, A\}_{K_B}$
6. $B \rightarrow A: \quad \{N\}_K$
7. $A \rightarrow B: \quad \{N-1\}_K$

A initiates the protocol by sending his identity to B. Once B receives the message, he replies with an encryption of whatever he received and a newly generated nonce N_B. A then forwards this encryption together with the identities and a new nonce N_A to the server S. S receives this message, decrypts the messages using the shared key K_B and responds by sending the new session key K encrypted for A with K_A in step 4 including a segment encrypted for B containing the key also. On receipt of the message in step 4, A decrypts it and checks the value of N_A to ensure that the key K is fresh. He then forwards the encrypted segment to B in step 5. B does a similar thing that decrypts the message and checks the freshness of the key K. In step 6 B challenge A with another nonce N. If A responds by sending the value decremented by one in step 7, the protocol is complete.

However, the protocol is vulnerable to a complex type flaw attack, discovered by B. W. Long [59]:

1. $A \rightarrow B: \quad A$
2. $B \rightarrow A: \quad \{A, N_B\}_{K_B}$
3. $A \rightarrow S: \quad A, B, N_A, \{A, N_B\}_{K_B}$

$1'$. $M \rightarrow A: \quad N_A, B, K'$

$2'$. $A \rightarrow M: \quad \{N_A, B, K', N'_A\}_{K_A}$

4. $M \rightarrow A: \quad \{N_A, B, K', N'_A\}_{K_A}$
5. $A \rightarrow B: \quad N'_A$
6. $M \rightarrow A: \quad \{N\}_{K'}$
7. $A \rightarrow M: \quad \{N-1\}_{K'}$

The attack requires two instances of the protocol, in which A plays the roles of initiator and responder, respectively. In the first instance, A initiates the protocol with B. Meanwhile, the attacker, M, initiates the second instance with A and sends N_A, B, K', in step $1'$, to A, where N_A is a copy from step 3 in the first instance and K' is a faked key generated by the attacker. A will generate and send out the encryption of whatever he received, N_A, B, K' in fact,

and a nonce N'_A. The attacker impersonates S and replays this message to A in the first instance. A decrypts this message, checks the nonce N_A and the identity B, and accepts K' as the session key, which is actually generated by the attacker. After completing the challenge and response in step 6 and 7, A will communicate with the attacker with the faked key K'.

2.4.1.4 Otway-Rees Protocol

The Otway-Rees protocol aims at establishing a fresh shared key, K, and distribute it between two principals A and B. The key establishment and distribution are done with the help of a server S. It is assumed that initially A and B share long term keys K_A and K_B with the server, respectively. The protocol is listed below:

1. $A \rightarrow B: \quad M, A, B, \{M, A, B, N_A\}_{K_A}$
2. $B \rightarrow S: \quad M, A, B, \{M, A, B, N_A\}_{K_A}, \{M, A, B, N_B\}_{K_B}$
3. $S \rightarrow B: \quad M, \{N_A, K\}_{K_A}, \{N_B, K\}_{K_B}$
4. $B \rightarrow A: \quad M, \{N_A, K\}_{K_A}$
5. $B \rightarrow A: \quad \{msg\}_K$

The value M is used as a serial number and provides no security intention, therefore we can safely assume that it is known to all the principals and even the attacker. The nonces N_A and N_B, on the other hand, is freshly generated in each session and thus should not be known to the attacker in advance. Note that we rearrange the orders of some elements, compared to the original protocol narration, just to enable pattern matching to work when encoding it in LySa.

1. Principal A generates a fresh nonce N_A, encrypts it along with the serial number M and the names of the principals and sends the encryption as well as the other information to the principal B.

2. Principal B generates another fresh nonce N_B, encrypts it with the value M and the names of the principals using the shared key and sends it together with what he received to the server S.

3. Server S generates a fresh session key, K, encrypts it with the nonces that is known to him after decrypting what he receives, using the long term keys, K_A and K_B, respectively. Along with the value M, the two encryptions are sent to B.

4. Principal B decrypts the last part of the message he receives using his long term key K_B and checks whether the nonce N_B is indeed the one he newly generated and sent out. If this is the case, he then accepts K as the new session key and forwards the rest of the message to principal A. Principal A also checks the nonce N_A and decides whether he accepts K as the session key.

5. Principal B and A now are communicating with each other using the key K to encrypt the messages.

2.4.1.5 Woo and Lam Protocol π_1

Woo and Lam [83] introduced a protocol that ensures one-way authentication of the initiator of the protocol, A, to a responder, B. The protocol uses symmetric-key cryptography and a trusted third-party server, S, with whom A and B share long-term symmetric keys. The protocol uses a fresh and unpredictable nonce N_B produced by B. The protocol narration is listed in the left part of the figure below, where the keys K_{AS} and K_{BS} represent the long-term keys that A and B share with the trusted server S. The protocol narration is the following,

$$
\begin{array}{lll}
1. & A \rightarrow B: & A \\
2. & B \rightarrow A: & N_B \\
3. & A \rightarrow B: & \{A, B, N_B\}_{K_{AS}} \\
4. & B \rightarrow S: & \{A, B, \{A, B, N_B\}_{K_{AS}}\}_{K_{BS}} \\
5. & S \rightarrow B: & \{A, B, N_B\}_{K_{BS}}
\end{array}
$$

The Woo-Lam protocol is prone to a type flaw attack, which is shown below. The attacker replays the nonce N_B to B in step 3, which B accepts as being of the form $\{A, B, N_B\}_{K_{AS}}$. B then encrypts whatever he received and then sends it out in step 4. The attacker intercepts it and replays it to B in step 5 and therefore fools B to believe that he has authenticated A, whereas A has not even participated in the run, as shown below,

$$
\begin{array}{lll}
1. & M(A) \rightarrow B: & A \\
2. & B \rightarrow M(A): & N_B \\
3. & M(A) \rightarrow B: & N_B \\
4. & B \rightarrow M(S): & \{A, B, N_B\}_{K_{BS}} \\
5. & M(S) \rightarrow B: & \{A, B, N_B\}_{K_{BS}}
\end{array}
$$

2.5 Why Cryptographic Protocols Go Wrong

Nowadays a considerable number of cryptographic protocols have been specified and implemented. Consequently analysing cryptographic protocols in order to find various kinds of flaws and to prevent them has received a lot of attention. The area is, however, remarkably subtle and a very large portion of proposed protocols have been shown to be flawed a long time after they were published. This has naturally encouraged research in this area. To give some examples, the Needham Schroeder Conventional Key Protocol was published in 1978 [70] and became the basis for many similar protocols in later years. In 1981, Denning and Sacco demonstrated that the protocol was flawed and subject to replay attack [29]. This sets the general trend for the field. In 1994, 13 years after the public protocol of Denning and Sacco has been published, Martin Abadi demonstrated that it was flawed [4]. 17 years after the publication, the public key protocol of Needham and Schroeder, in 1995, was proved to be flawed by Lowe. In the intervening years a whole host of protocols have been specified and found to be flawed.

One reason for cryptographic protocols easily going wrong is the existence of the attacker. As mentioned before, cryptographic protocols are deployed over an open network such that everyone can join it and start sending and receiving messages to and from the principals across it without the need of authorization or permission. In such an open environment, we mush anticipate that there are bad guys out there who will do all sorts of bad things, not just passively eavesdropping, but also actively altering, forging, duplicating, re-directing, deleting or injecting messages. These fault messages can be malicious and cause a destructive effect to the principals in the receiving end.

Thus, in the Dolev-Yao threat model, any message sent to the network is considered to be sent to the attacker for his disposal. Consequently, any message received from the network is treated to have been received from the attacker after his disposal. In other words, the attacker is considered to have the complete control of the entire network.

Another reason for the cryptographic protocols easily go wrong is that, in the cryptographic protocol literature, protocols are usually expressed as narrations and most of the details of the actual deployment are ignored. Encryption, decryption and other operations are modelled using only the natural algebraic laws they always obey. All data that is exchanged in a cryptographic protocol is presented in symbolic form.

Despite being rather intuitive, the description technique of protocol narrations contains lots of implicit concepts. Abadi [2] pointed out that “informal protocol

narrations" need to be complemented with explanations of some either implicitly assumed facts or additional information to remove ambiguities. For example: (1) one should make explicit what is known (publicly and privately) before a protocol run, and what is to be generated freshly during a protocol run, (2) one should make explicit what security goals the protocol is assumed to provide. In case of authentication, both principals involved should be convinced that the session key is known by both of them only but nobody else (except for the trusted server). In case of key freshness, both principals should be convinced that the session key is a newly generated one. These are exactly the cases for the Needham-Schroeder symmetric key protocol in Example 2.5.

All the above points seem to be suggesting further work to make protocol narrations more explicitly and formally. However, instead of pursuing a formal semantics for protocol narrations, we shall introduce, in next chapter, some other ways of describing protocols, namely process calculi, which have complete semantics greatly facilitating both modelling and statically analysing cryptographic protocols.

Chapter 3

The LySa Calculus for Security Protocols

In the literature, process calculus has the central position as a framework of modelling and reasoning about concurrent systems. In the last two decades several variants of process calculus have come into existence, the most important ones of which include Communicating Sequential Processes (CSP) by Hoare [50, 51], the Calculus of Communicating Systems (CCS) by Milner [67, 68], and the Algebra of Communicating Processes (ACP) by Bergstra and Klop [8, 10, 11]. Initially, the main idea of process calculi is to have a simple language and is focused on studying the communications in a fixed network of parallel processes. Later on, the use of processes calculi was evolved into describing dynamically reconfigurable system, e.g. the π calculus [66] and Mobile Ambients [25], or focusing on a particular aspect of a system, such as real-time [27, 49, 69, 85], probabilistic [39, 40, 47] or security features [3, 15].

Process calculi offer a pure framework to study concurrent and distributed systems and, in turn, the security issues connected to them. Systems are specified as expressions of the calculus, called *processes*. Processes are obtained by combining via a few operators (sequential and parallel composition, nondeterministic choice, declarations), and can perform actions like sending and receiving messages along channels. Furthermore there are some scope operators, such as restrictions and hiding.

Every process calculus is supplied with an operational semantics, which is based on transition systems in most of the cases. The operational semantics concerns the process behaviour. It describes which activities and operational steps a process can perform.

3.1 LySA Calculus

In this section we shall briefly recall the LySA [15, 16, 20] calculus. The process calculus LySA is designed specially for modelling security protocols. It is in the π-calculus tradition and develops the idea from the Spi-calculus for incorporation of cryptographic operations. However different from both $\pi-$ and Spi-calculus, it has the following two features, which greatly facilitate developing the static analysis (see next chapter).

Firstly, LySA has no communication channels and instead all the communication takes place on a global network, ether. This corresponds to the real scenario where typically security protocols are operated.

Secondly, opposing to have a separate matching construct, e.g. if-then construct in some other process calculi, LySA directly incorporates pattern matching into the language constructs where values can become bound to variables. This not only makes modelling of protocols more succinct but also makes the analysis simpler, because one does not have to deal with values that have become bound in a early place and will be filtered by matching in a later place.

3.1.1 Syntax

In LySA, the basic building block is values, i.e. Val, which are used to represent keys, nonces, encrypted messages, etc. Syntactically, they are described by expressions $E \in Expr$ that may either be variables, names, or encryptions. Variables and names come from two disjoint sets Var, ranged over by x, and $Name$, respectively. The set $Name$ can be further partitioned into two sets: ordinary names, representing principal names, nonces and symmetric keys, which are ranged over by n, and key pairs, m^+ and m^-, representing public and private key pairs for asymmetric key cryptography. Finally, expressions may be encryptions of a k-tuple of other expressions. Both symmetric key encryption $\{E_1, \ldots, E_k\}_{E_0}$, and asymmetric key encryption $\{|E_1, \ldots, E_k|\}_{E_0}$ can be modelled in LySA. In both cases, E_0 is the key used to perform the encryption.

LySA expressions are, in turn, used to construct LySA processes $P \in Proc$ as shown in Table 3.1.

$E ::=$	*terms*
n	name
x	variable
m^+	public key
m^-	private key
$\{E_1, \ldots, E_k\}_{E_0}$	symmetric encryption
$\{\!\|E_1, \ldots, E_k\|\!\}_{E_0}$	asymmetric encryption
$P ::=$	*processes*
$\langle E_1, \ldots, E_k \rangle.P$	output
$(E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P$	input
$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}_{E_0}\ \mathsf{in}\ P$	symmetric decryption
$\mathsf{decrypt}\ E\ \mathsf{as}\ \{\!\|E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\|\!\}_{E_0}\ \mathsf{in}\ P$	asymmetric decryption
$(\nu\ n)P$	restriction
$(\nu_{\pm}\ m)P$	pair restriction
$P_1 \mid P_2$	parallel composition
$!P$	replication
0	nil

Table 3.1: Syntax of LySA calculus

The process $\langle E_1, \ldots, E_k \rangle.P$ sends a $k-$tuple of values onto the global network and, if it succeeds, continues as P.

The process $(E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P$ reads a $k-$tuple of values from the global network. Input incorporates pattern matching and only successfully continues as P when the matching succeeds, namely, the first j values of the received k-tuple are pair-wise equal to $E_1, \ldots, E_j$. In this case, the rest $k-j$ values of the received tuple are pair-wise bound to the variables $x_{j+1}, \ldots, x_k$. Syntactically, the expressions used for matching and the variables to be bound are separated by a semi-colon.

The process $\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}_{E_0}\ \mathsf{in}\ P$ denotes the symmetric decryption. Similar to the input construct, it incorporates a form of pattern matching, too. The decryption succeeds only when the value of term E is an encryption of $k-$tuple, where the first j values are pair-wise identical to $E_1, \ldots, E_j$ and furthermore the encryption key has to be identical to E_0. If this is the case, the rest $k-j$ values are pair-wise bound to the variables $x_{j+1}, \ldots, x_k$ and the process continues as P.

The process decrypt E as $\{|E_1, \ldots, E_j; x_{j+1}, \ldots, x_k|\}_{E_0}$ in P denotes the asymmetric decryption. It is the same as symmetric decryption except that the encryption key and E_0 must from a key pair, i.e. m^+ and m^-. It does not matter which is the public one and which is the private one. In this way LySA is able to model both public key encryption and private key signatures.

The process $(\nu\ n)P$ generates a fresh name n, of which the scope is restricted to be the process P.

The process $(\nu_{\pm}\ m)P$, similar to $(\nu\ n)P$, generates a pair of fresh names m^+ and m^-. Their scopes are restricted to be P, too.

The process $P_1 \mid P_2$ denotes two processes P_1 and P_2 running in parallel. They may synchronise through communication over the global network or perform internal actions independently.

$!P$ generates an arbitrary number of process P composed in parallel.

0 is the nil process and does nothing.

Example 3.1 *The protocol in Example 2.4 can be modelled in LySA as the following process.*

$$
\begin{array}{l}
(\nu\ K)(\\
\quad (\nu\ CCI)\langle A, B, \{CCI\}_K\rangle.0 \\
\quad \mid \\
\quad (A, B; y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI\}_K\ \mathsf{in}\ 0)
\end{array}
$$

The process shows a communication between two principals, namely A and B. The topmost and bottommost processes in the parallel composition represent their actions, respectively. Initially, the principals A and B are assumed to share a symmetric key K.

In LySA, the prefix $(\nu\ K)$ in the process $(\nu\ K)P$ is called a *binder* of the name K, which has the scope of the process P. Also the process $(\nu\ CCI)P'$ is a *binder* of name CCI, restricting its scope to the process P'. A name is said to be *free*, whenever its occurrence is not *bound* by any binder. In LySA, a standard function $fn(P)$ is defined, which collects all the free names in the process P. This function is straightforward and standard but for completeness it is listed in Table 3.2.

In the protocols, first, the principal A generates a fresh value CCI, for example his credit card information, and then out puts the message $\langle A, B, \{CCI\}_K\rangle$,

consisting the names of the sender A and the intended receiver B and finally the credit card information encrypted under the shared symmetric key, $\{CCI\}_K$, onto the network.

The principal B is willing to receive a triple sent to himself. He uses pattern matching to ensure that the first two values are indeed A, B. On successfully receiving the triple sent by A, the variable y is bound to the value $\{CCI\}_K$. In order to get the credit card information, the principal B has to decrypt the value using the key K. If succeeds, the variable $yCCI$ will be then bound to the value CCI.

The two parallel processes are terminated by the inactive process.

Example 3.2 *The process below represents a protocol between two principals A and B. It is similar to the one in Example 3.1 but different in the choosing of asymmetric key cryptography.*

$$!(\nu\ CCI)(B, A; x).\langle A, B, \{CCI\}_x\rangle.0$$
$$|$$
$$!(\nu_{\pm}\ K)\langle B, A, K^+\rangle.(A, B; y).\textsf{decrypt}\ y\ \textsf{as}\ \{; yCCI\}_{K^-}\ \textsf{in}\ 0$$

First, principal B generates a fresh key pair K^+ and K^-. It sends K^+ to the principal A while K^- is kept private to the principal A due to the scoping rule of the pair restriction operator.

On reception, the principal A encrypts his credit card information CCI using the public key received in the variable x. This message is sent to B that decrypts it using the private key K^-. On successful decryption B has the credit card information stored in the variable $yCCI$.

3.1.2 Semantics

Following the π-calculus tradition, LySA has a reduction semantics. The reduction relation holds between a pair of processes, written as $P \rightarrow P'$, meaning that P can evolve into P'. The definition of reduction relation is described by axioms and inference rules that form an inductive definition of the relation. Before moving to the definition of the reduction relation itself in Table 3.5, some auxiliary mechanisms will be explained.

$fn(n)$	$\stackrel{def}{=}$	$\{n\}$
$fn(x)$	$\stackrel{def}{=}$	$\emptyset$
$fn(\{E_1, \ldots, E_k\}_{E_0})$	$\stackrel{def}{=}$	$fn(E_0) \cup \ldots \cup fn(E_k)$
$fn(\{\!\|E_1, \ldots, E_k\|\!\}_{E_0}$	$\stackrel{def}{=}$	$fn(E_0) \cup \ldots \cup fn(E_k)$
$fn(\langle E_1, \ldots, E_k \rangle.P)$	$\stackrel{def}{=}$	$fn(E_1) \cup \ldots \cup fn(E_k) \cup fn(P)$
$fn((E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P)$	$\stackrel{def}{=}$	$fn(E_1) \cup \ldots \cup fn(E_j) \cup fn(P)$
$fn(\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}_{E_0}\ \mathsf{in}\ P)$	$\stackrel{def}{=}$	$fn(E) \cup fn(E_0) \cup \ldots \cup fn(E_j) \cup fn(P)$
$fn(\mathsf{decrypt}\ E\ \mathsf{as}\ \{\!\|E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\|\!\}_{E_0}\ \mathsf{in}\ P)$	$\stackrel{def}{=}$	$fn(E) \cup fn(E_0) \cup \ldots \cup fn(E_j) \cup fn(P)$
$fn((\nu\ n)P)$	$\stackrel{def}{=}$	$fn(P) \setminus \{n\}$
$fn((\nu_{\pm}\ m)P)$	$\stackrel{def}{=}$	$fn(P) \setminus \{m^+, m^-\}$
$fn(P_1 \mid P_2)$	$\stackrel{def}{=}$	$fn(P_1) \cup fn(P_2)$
$fn(!P)$	$\stackrel{def}{=}$	$fn(P)$
$fn(0)$	$\stackrel{def}{=}$	$\emptyset$

Table 3.2: Free names; $fn(P)$

3.1.2.1 Structural Congruence

The *structural congruence* $\equiv$ is defined as the least congruence satisfying the following conditions. The idea is that two processes are considered to be equal when they are identical up to structure and only differ in their syntax.

The last rule in Table 3.3 says that two processes P_1 and P_2 are structurally equivalent whenever they are $\alpha-$equivalent, namely $P_1 \stackrel{\alpha}{\equiv} P_2$. The $\alpha-$equivalence relation is used to express the idea that the names of the bound variables are unimportant and can be substituted by another one (under certain conditions). The rules defining $\alpha-$equivalence are listed in Table 3.4.

$$
\begin{array}{l}
P \equiv P \\
P_1 \equiv P_2 \text{ implies } P_2 \equiv P_1 \\
P_1 \equiv P_2 \text{ and } P_2 \equiv P_3 \text{ implies } P_1 \equiv P_3 \\
P_1 \equiv P_2 \text{ implies } \left\{ \begin{array}{l}
\langle E_1, \ldots, E_k \rangle.P_1 \equiv \langle E_1, \ldots, E_k \rangle.P_2 \\
(E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P_1 \equiv \\
\quad (E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P_2 \\
\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}_{E_0}\ \mathsf{in}\ P_1 \equiv \\
\quad \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}_{E_0}\ \mathsf{in}\ P_2 \\
\mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j; x_{j+1}, \ldots, x_k|\!\}_{E_0}\ \mathsf{in}\ P_1 \equiv \\
\quad \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j; x_{j+1}, \ldots, x_k|\!\}_{E_0}\ \mathsf{in}\ P_2 \\
(\nu\ n)P_1 \equiv (\nu\ n)P_2 \\
(\nu_{\pm}\ m)P_1 \equiv (\nu_{\pm}\ m)P_2 \\
P_1 \mid P_3 \equiv P_2 \mid P_3 \\
!P_1 \equiv !P_2
\end{array} \right. \\
P_1 \mid P_2 \equiv P_2 \mid P_1 \\
(P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3) \\
P \mid 0 \equiv P \\
!P \equiv P \mid !P \\
(\nu\ n)0 \equiv 0 \\
(\nu\ n_1)(\nu\ n_2)P \equiv (\nu\ n_2)(\nu\ n_1)P \\
(\nu\ n)(P_1 \mid P_2) \equiv P_1 \mid (\nu\ n)P_2 \text{ if } n \notin fn(P_1) \\
(\nu_{\pm}\ m)0 \equiv 0 \\
(\nu_{\pm}\ m_1)(\nu_{\pm}\ m_2)P \equiv (\nu_{\pm}\ m_2)(\nu_{\pm}\ m_1)P \\
(\nu_{\pm}\ m)(P_1 \mid P_2) \equiv P_1 \mid (\nu_{\pm}\ m)P_2 \text{ if } m^+, m^- \notin fn(P_1) \\
(\nu_{\pm}\ m)(\nu\ n)P \equiv (\nu\ n)(\nu_{\pm}\ m)P \\
P_1 \overset{\alpha}{\equiv} P_2 \text{ implies } P_1 \equiv P_2
\end{array}
$$

Table 3.3: Structural congruence; $P \equiv P'$

3.1.2.2 The Reduction Relation

The reduction relation, describing how a process evolves into another, is defined inductively as the smallest relation on two processes such that the rules in Table 3.5 are satisfied.

One auxiliary ingredient in the definition of the reduction relation is substitution of variables for values. The values $V \in Val$ are simply the closed expressions, i.e.

$P \overset{\alpha}{\equiv} P$	
$P_1 \overset{\alpha}{\equiv} P_2$ implies $P_2 \overset{\alpha}{\equiv} P_1$	
$P_1 \overset{\alpha}{\equiv} P_2$ and $P_2 \overset{\alpha}{\equiv} P_3$ implies $P_1 \overset{\alpha}{\equiv} P_3$	
$(\nu\ n_1)P \overset{\alpha}{\equiv} (\nu\ n_2)(P[n_1 \mapsto n_2])$	if $n_2 \notin fn(P)$
$(\nu_{\pm}\ m_1)P \overset{\alpha}{\equiv} (\nu_{\pm}\ m_2)(P[m_1^+ \mapsto m_2^+, m_1^- \mapsto m_2^-])$	if $m_2^+, m_2^- \notin fn(P)$

Table 3.4: $\alpha-$equivalence

expressions without variables. Formally, values may be built from the grammar:

$$\begin{array}{rcl} V & ::= & n \\ & | & m^+ \\ & | & m^- \\ & | & \{V_1, \ldots, V_k\}_{V_0} \\ & | & \{\!|V_1, \ldots, V_k|\!\}_{V_0} \end{array}$$

The semantics of the calculus specifies how a process may evolve. Because of the parallelism construct, a process may evolve in various ways. One may be interested in whether it is the expected one. To this end, the semantics of LySA makes use of a reference monitor to check each computational steps against certain additional requirements before allowing it to be executed. The reference monitor can be either turned on or off, thus gives two variants, parameterised on $\mathcal{R}$.

1. the standard semantics, written as $P \rightarrow P'$, takes $\mathcal{R}$ to be universally true and thus ignores it. This can be viewed as that the reference monitor has been turned off, i.e. no additional requirement has to be meet.

2. the reference monitor semantics, written as $P \rightarrow_{\mathsf{RM}} P'$, is an extension of the standard semantics. In this case, the reference monitor is turned on and checks properties of a process at run time. If the requirements, defined by RM, are not meet, the process execution will be aborted by the reference monitor. Note that RM is currently undefined, because no security properties have been taken into account yet. In later chapters, we shall show that how the reference monitor works dynamically to check various security properties.

In some of the reduction rules, a substitution function has been used, written as $P[V/x]$, which substitutes a variable x for a value V in the process P whenever x becomes bound to V.

$$\text{(Com)}\quad \frac{\wedge_{i=1}^{k} V_i = V_i'}{\langle V_1, \ldots, V_k\rangle.P \mid (V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k).P' \rightarrow_{\mathcal{R}} P \mid P'[V_{j+1}'/x_{j+1}, \ldots, V_k'/x_k]}$$

$$\text{(Dec)}\quad \frac{\wedge_{i=0}^{k} V_i = V_i'}{\mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}\ \mathsf{as}\ \{V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k\}_{V_0'}\ \mathsf{in}\ P \rightarrow_{\mathcal{R}} P[V_{j+1}'/x_{j+1}, \ldots, V_k'/x_k]}$$

$$\text{(ADec)}\quad \frac{\wedge_{i=1}^{k} V_i = V_i'}{\mathsf{decrypt}\ \{\!|V_1, \ldots, V_k|\!\}_{m^+}\ \mathsf{as}\ \{\!|V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k|\!\}_{m^-}\ \mathsf{in}\ P \rightarrow_{\mathcal{R}} P[V_{j+1}'/x_{j+1}, \ldots, V_k'/x_k]}$$

$$\text{(ASig)}\quad \frac{\wedge_{i=1}^{k} V_i = V_i'}{\mathsf{decrypt}\ \{\!|V_1, \ldots, V_k|\!\}_{m^-}\ \mathsf{as}\ \{\!|V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k|\!\}_{m^+}\ \mathsf{in}\ P \rightarrow_{\mathcal{R}} P[V_{j+1}'/x_{j+1}, \ldots, V_k'/x_k]}$$

$$\text{(New)}\quad \frac{P \rightarrow_{\mathcal{R}} P'}{(\nu\ n)P \rightarrow_{\mathcal{R}} (\nu\ n)P'} \qquad \text{(ANew)}\quad \frac{P \rightarrow_{\mathcal{R}} P'}{(\nu_{\pm}\ m)P \rightarrow_{\mathcal{R}} (\nu_{\pm}\ m)P'}$$

$$\text{(Par)}\quad \frac{P_1 \rightarrow_{\mathcal{R}} P_1'}{P_1 \mid P_2 \rightarrow_{\mathcal{R}} P_1' \mid P_2} \qquad \text{(Congr)}\quad \frac{P \equiv P' \ \wedge\ P' \rightarrow_{\mathcal{R}} P'' \wedge P'' \equiv P'''}{P \rightarrow_{\mathcal{R}} P'''}$$

Table 3.5: Operational Semantics; $P \rightarrow_{\mathcal{R}} P'$.

The reduction relation itself is defined inductively as the smallest relation on pairs of processes that satisfies the rules in Table 3.5. These rules are explained below.

The rule (Com) expresses that a communication happens between two processes only when there is an output $\langle V_1, \ldots, V_k\rangle.P$ and an input $(V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k).P'$, where the first j values $V_1, \ldots, V_j$ and $V_1', \ldots, V_j'$ are pairwise identical. In this case, the variables after the semicolon in the input construct are substituted for the corresponding values in the output construct.

The rules (Dec), (ADec) and (ASig) concern about decryptions and perform pattern matching in a similar way: it is required that the expression being decrypted is an encryption value of the right form, i.e. the first j values are $V_1, \ldots, V_j$, and the keys are the right ones with respect to symmetric or asymmetric key cryptography. If these requirements are met, variables are substituted for the corresponding values.

The rule (New) and (ANew) let a process to evolve inside a restriction, however the restriction operator itself does never disappear.

The rule (Par) expresses that parallel composition is interleaved such that one of its branches may evolve while the other one remains unchanged.

Finally, the rule (Congr) ensures that the reduction relation may be applied to any process that is structurally congruent to the processes found in the other rules.

Example 3.3 *Consider the process from Example 3.1. According to the reduction relation rules in Table 3.5, it may evolve following the steps:*

$$
\begin{array}{ll}
 & (\nu\ K)((\nu\ CCI)\langle A, B, \{CCI\}_K\rangle.0 \mid (A, B; y).\textsf{decrypt}\ y\ \textsf{as}\ \{; yCCI\}_K\ \textsf{in}\ 0) \\
\rightarrow & (\nu\ CCI)\langle A, B, \{CCI\}_K\rangle.0 \mid (A, B; y).\textsf{decrypt}\ y\ \textsf{as}\ \{; yCCI\}_K\ \textsf{in}\ 0 \\
\rightarrow & \langle A, B, \{CCI\}_K\rangle.0 \mid (A, B; y).\textsf{decrypt}\ y\ \textsf{as}\ \{; yCCI\}_K\ \textsf{in}\ 0 \\
\rightarrow & 0 \mid \textsf{decrypt}\ \{CCI\}_K\ \textsf{as}\ \{; yCCI\}_K\ \textsf{in}\ 0 \\
\rightarrow & 0 \mid 0
\end{array}
$$

Note that in the line 3, since the output of the principal A, $\langle A, B, \{CCI\}_K\rangle$ and the input of the principal B, $(A, B; y)$, match each other, the variable y is then substituted for the value $\{CCI\}_K$ in the rest of the process and hence gives decrypt $\{CCI\}_K$ as $\{; yCCI\}_K$ in 0 *in the line 4. In the line 5, we write $0 \mid 0$ as a shorthand for $0 \mid 0[CCI/yCCI]$, which, indeed, are equivalent.*

Example 3.4 *In pattern matching, values to be matched and variables to be bound are separated by semicolon. Thus the location of the semicolon is important and affects how a process may evolve. Consider the following two cases:*

$$1) \quad \langle n\rangle.P \mid (; m).Q \rightarrow P|Q[n/m]$$

m is a variable and the process progresses.

$$2) \quad \langle n\rangle.P \mid (m;).Q$$

m is a value and the process is stuck.

3.2 The Meta Level Calculus

Each protocol can be used in a variety of scenarios to meet different needs, which are not necessarily known in the design stage. When it comes to the deployment stage, very often, one may need to make a number of assumptions, e.g. how

many principals will be executing the protocol simultaneously and whether a principal is allowed to play different roles at the same time. However, the LySA calculus presented in the previous section is abstracted to a higher level such that it is not capable of describing those assumptions clearly. To this aim, a *meta level* is introduced as an extension of LySA presented before, which, from now on, is referred to as *object level* for distinguishment.

Basically, the meta level is developed for describing different scenarios when many principals execute a protocol at the same time, i.e. from the protocol's point of view, there are many initiators and responders. To identify those principals, a protocol can be modelled as several copies of processes representing each principal and renaming each individual principal names and keys to be unique. This amounts to syntactically unfolding a process, e.g. using the rule $!P \equiv P \mid !P$, and attaching indices to each copy.

The meta level is an extension of the object level with adding sequence of indices, $i_1 \ldots i_k$, to names and variables. Usually we write $\bar{i}$ as a shorthand of $i_1 \ldots i_k$. The syntax of meta level terms $ME \in MTerm$ and meta level processes $MP \in MProc$ are defined by the grammar in Table 3.6:

The meta level processes have three new constructs. They all incorporate countable indexing sets S, which includes set variables X.

- $|_{i \in S} MP$ is the parallel composition of instances of process MP, where the index i throughout each instance is substituted by a element in the set S.
- $\mathsf{let}\ X \subseteq S\ \mathsf{in}\ MP$ declares a set identifier X to be used in the process M, where $X \in SetId$ refers to a subset of the values of the index set S in the process MP. For the reason of consistence, whenever X is instantiated to a subset of S, the same instantiation will be applied to all the X occurring throughout the process M.
- $(\nu_{\bar{i} \in \bar{S}}\ n_{\bar{a}\bar{i}})MP$ is a restriction of all names $n_{\bar{a}\bar{i}}$, where the possible empty prefix of indices $\bar{a}$ have already been defined and the index sequence $\bar{i} = i_1 \ldots i_k$ is instantiated pairwisely from the set $\bar{S} = S_1 \times \ldots \times S_k$.
- $(\nu_{\pm \bar{i} \in \bar{S}}\ m_{\bar{a}\bar{i}})MP$ is as above except that it restricts the pairs of indexed names $m^{+}_{\bar{a}\bar{i}}$ and $m^{-}_{\bar{a}\bar{i}}$.

Example 3.5 *Consider the process from Example 3.1. It can be deployed in a scenario where there are many principals A_i for i in some set S_1 and many B_j for j in some set S_2. This scenario can be modelled by a meta level process as:*

$$
\begin{array}{lcl}
mx & ::= & x_{\tilde{i}} \\
\\
ME & ::= & n_{\tilde{i}} \\
 & | & mx \\
 & | & m^{+}_{\tilde{i}} \\
 & | & m^{-}_{\tilde{i}} \\
 & | & \{ME_1, \ldots, ME_k\}_{ME_0} \\
 & | & \{\!|ME_1, \ldots, ME_k|\!\}_{ME_0} \\
\\
MP & ::= & |_{i \in S} MP \\
 & | & \mathsf{let}\ X \subseteq S\ \mathsf{in}\ MP \\
 & | & (\nu_{\tilde{i} \in \bar{S}}\ n_{\bar{a}\tilde{i}}) MP \\
 & | & (\nu_{\pm\tilde{i} \in \bar{S}}\ m_{\bar{a}\tilde{i}}) MP \\
 & | & \langle ME_1, \ldots, ME_k \rangle.MP \\
 & | & (ME_1, \ldots, ME_j; mx_{j+1}, \ldots, mx_k).MP \\
 & | & \mathsf{decrypt}\ ME\ \mathsf{as}\ \{ME_1, \ldots, ME_j; mx_{j+1}, \ldots, mx_k\}_{ME_0}\ \mathsf{in}\ MP \\
 & | & \mathsf{decrypt}\ ME\ \mathsf{as}\ \{\!|ME_1, \ldots, ME_j; mx_{j+1}, \ldots, mx_k|\!\}_{ME_0}\ \mathsf{in}\ MP \\
 & | & (\nu\ n_{\tilde{i}}) MP \\
 & | & (\nu_{\pm}\ m_{\tilde{i}}) MP \\
 & | & MP_1 \mid MP_2 \\
 & | & !MP \\
 & | & 0
\end{array}
$$

Table 3.6: Syntax of meta level LySA calculus

$$
\begin{array}{l}
\mathit{let}\ X \subseteq S_1\ \mathit{in} \\
\mathit{let}\ Y \subseteq S_2\ \mathit{in} \\
(\nu\ K)(\\
\quad |_{i \in X}|_{j \in Y} (\nu\ CCI_{ij}) \langle A_i, B_j, \{CCI_{ij}\}_K \rangle.0 \\
| \\
\quad |_{i \in X}|_{j \in Y} (A_i, B_j; y_{ij}).\mathit{decrypt}\ y_{ij}\ \mathit{as}\ \{; yCCI_{ij}\}_K\ \mathit{in}\ 0)
\end{array}
$$

In this scenario, each principal A_i initiates a session with each principal B_j and all the principals share the same key K.

The meta level is an extension of the object level by attaching indies to variables and names. By instantiating the indies, it generates a number of object level processes, each of which represents a instance of the deployment scenario. This instantiation relation is formally defined as $MP \Rrightarrow P$, describing that an object level process P is an instance of a meta level process MP. The instantiation relation is defined in Table 3.7.

(ILet)	$\dfrac{MP[X \mapsto S'] \Rrightarrow P}{\mathsf{let}\ X \subseteq S\ \mathsf{in}\ MP \Rrightarrow P} \quad \text{if } S' \subseteq_{fin} S$
(IIPar)	$\dfrac{MP[i \mapsto a_1] \Rrightarrow P_1 \quad \ldots \quad MP[i \mapsto a_k] \Rrightarrow P_k}{\mid_{i \in \{a_1,\ldots,a_k\}} MP \Rrightarrow P_1 \mid \ldots \mid P_k}$
(IINew)	$\dfrac{MP \Rrightarrow P}{(\nu_{\bar{i} \in \{\overline{a_1},\ldots,\overline{a_k}\}}\ n_{\bar{a}\bar{i}})MP \Rrightarrow (\nu\ n_{\overline{aa_1}}) \ldots (\nu\ n_{\overline{aa_k}})P}$
(IIANew)	$\dfrac{MP \Rrightarrow P}{(\nu_{\pm\bar{i} \in \{\overline{a_1},\ldots,\overline{a_k}\}}\ m_{\bar{a}\bar{i}})MP \Rrightarrow (\nu_{\pm}\ m_{\overline{aa_1}}) \ldots (\nu_{\pm}\ m_{\overline{aa_k}})P}$
(IOut)	$\dfrac{MP \Rrightarrow P}{\langle ME_1,\ldots,ME_k\rangle.MP \Rrightarrow \langle ME_1,\ldots,ME_k\rangle.P}$
(IInp)	$\dfrac{MP \Rrightarrow P}{(ME_1,\ldots,ME_j; mx_{j+1},\ldots,mx_k).MP \Rrightarrow (ME_1,\ldots,ME_j; mx_{j+1},\ldots,mx_k).P}$
(IDec)	$\dfrac{MP \Rrightarrow P}{\mathsf{decrypt}\ ME\ \mathsf{as}\ \{ME_1,\ldots,ME_j; mx_{j+1},\ldots,mx_k\}_{ME_0}\ \mathsf{in}\ MP \Rrightarrow \mathsf{decrypt}\ ME\ \mathsf{as}\ \{ME_1,\ldots,ME_j; mx_{j+1},\ldots,mx_k\}_{ME_0}\ \mathsf{in}\ P}$
(IADec)	$\dfrac{MP \Rrightarrow P}{\mathsf{decrypt}\ ME\ \mathsf{as}\ \{\mid ME_1,\ldots,ME_j; mx_{j+1},\ldots,mx_k\mid\}_{ME_0}\ \mathsf{in}\ MP \Rrightarrow \mathsf{decrypt}\ ME\ \mathsf{as}\ \{\mid ME_1,\ldots,ME_j; mx_{j+1},\ldots,mx_k\mid\}_{ME_0}\ \mathsf{in}\ P}$
(INew)	$\dfrac{MP \Rrightarrow P}{(\nu\ n_{\bar{a}})MP \Rrightarrow (\nu\ n_{\bar{a}})P}$
(IANew)	$\dfrac{MP \Rrightarrow P}{(\nu_{\pm}\ m_{\bar{a}})MP \Rrightarrow (\nu_{\pm}\ m_{\bar{a}})P}$
(IRep)	$\dfrac{MP \Rrightarrow P}{!MP \Rrightarrow !P}$
(IPar)	$\dfrac{MP_1 \Rrightarrow P_1 \quad MP_2 \Rrightarrow P_2}{MP_1 \mid MP_2 \Rrightarrow P_1 \mid P_2}$
(INil)	$0 \Rrightarrow 0$

Table 3.7: The instantiation relation; $MP \Rrightarrow P$

The rule (ILet) instantiates a meta level process M to all the object level process P, which can be found by taking some *finite* subset of the set S declared in the let construct. The operator $\subseteq_{fin}$ ensures that the subset taken is a finite one.

The rule (IIPar) instantiates an indexed parallel $|_{i \in S}\ M$ to be the parallel composition of processes for each of the indices in the set S.

The rules (IINew) and (IIANew) instantiate both of the indexed restrictions to the restrictions of the names for all values in the set $\{\overline{a_1}, \ldots, \overline{a_k}\}$.

Instantiation of all the other meta level constructs is performed simply by instantiating their subprocess. This is the case for the rest of the rules, i.e. (IOut), (IInp), (IDec), (IADec), (INew), (IANew), (IRep), (IPar) and (INil).

Example 3.6 *The meta level process from Example 3.4 can be instantiated to various different object level processes, when* $S_1 = \{1, 2\}$ *and* $S_1 = \{1\}$.

Taking $X = \{1, 2\}$ *and* $Y = \{1\}$*, we have:*

$$
\begin{array}{l}
(\nu\ K)(\\
\quad (\nu\ CCI_{11})\langle A_1, B_1, \{CCI_{11}\}_K\rangle.0 \\
\mid \quad (\nu\ CCI_{21})\langle A_2, B_1, \{CCI_{21}\}_K\rangle.0 \\
\mid \quad (A_1, B_1; y_{11}).\mathsf{decrypt}\ y_{11}\ \mathsf{as}\ \{; yCCI_{11}\}_K\ \mathsf{in}\ 0 \\
\mid \quad (A_2, B_1; y_{21}).\mathsf{decrypt}\ y_{21}\ \mathsf{as}\ \{; yCCI_{21}\}_K\ \mathsf{in}\ 0)
\end{array}
$$

Taking $X = \{1\}$ *and* $Y = \{1\}$*, we have:*

$$
\begin{array}{l}
(\nu\ K)(\\
\quad (\nu\ CCI_{11})\langle A_1, B_1, \{CCI_{11}\}_K\rangle.0 \\
\mid \quad (A_1, B_1; y_{11}).\mathsf{decrypt}\ y_{11}\ \mathsf{as}\ \{; yCCI_{11}\}_K\ \mathsf{in}\ 0)
\end{array}
$$

Taking $X = \emptyset$ *and* $Y = \emptyset$*, we have: 0.*

The (*ILet*) *rule in Table 3.7 decides that the meta level process instantiates to all the combinations of subsets of* S_1 *and* S_2*, which describes the scenario that at most two principals playing the role of initiator,* A_1 *and* A_2*, send a message to the responder* B_1.

3.3 A Worked Example: the Otway-Rees Protocol

When modelling a protocol in the meta level, there are a number of assumptions to make. The assumptions amounts to different scenarios, in which the protocol is deployed. In this section, we shall show some of the considerations when encoding a protocol by applying them to an example protocol, the Otway-Rees protocol, which has a narration as below and is explained in Chapter 2. We will show in the later chapters that the different assumptions made are crucial to the control flow analysis and may result in completely different analysis results.

1. $A \rightarrow B: \quad M, A, B, \{M, A, B, N_A\}_{K_A}$
2. $B \rightarrow S: \quad M, A, B, \{M, A, B, N_A\}_{K_A}, \{M, A, B, N_B\}_{K_B}$
3. $S \rightarrow B: \quad M, \{N_A, K\}_{K_A}, \{N_B, K\}_{K_B}$
4. $B \rightarrow A: \quad M, \{N_A, K\}_{K_A}$
5. $B \rightarrow A: \quad \{msg\}_K$

A very simple scenario of deploying the protocol is that only one principal A initiates a session with one principal B. These two principals and the server can be modelled as the three parallel processes listed in Table 3.8.

$$
\begin{array}{l}
(\nu\ K_A)(\nu\ K_B)(\\
\quad !(\nu\ N_A)\langle M, A, B, \{M, A, B, N_A\}_{K_A}\rangle. \\
\quad (M; xEnc).\mathsf{decrypt}\ xEnc\ \mathsf{as}\ \{N_A; xk\}_{K_A}\ \mathsf{in} \\
\quad (; xmEnc).\mathsf{decrypt}\ xmEnc\ \mathsf{as}\ \{; xMsg\}_{xk}\ \mathsf{in}\ 0 \\
\\
\mid\ !(M, A, B; yEnc). \\
\quad (\nu\ N_B)\langle M, A, B, yEnc, \{M, A, B, N_B\}_{K_B}\rangle. \\
\quad (M; yxEnc, yzEnc).\mathsf{decrypt}\ yzEnc\ \mathsf{as}\ \{N_B; yk\}_{K_B}\ \mathsf{in} \\
\quad \langle M, yxEnc\rangle.(\nu\ Msg)\langle\{Msg\}_{yk}\rangle.0 \\
\\
\mid\ !(M, A, B; zxEnc, zyEnc). \\
\quad \mathsf{decrypt}\ zxEnc\ \mathsf{as}\ \{M, A, B; zna\}_{K_A}\ \mathsf{in} \\
\quad \mathsf{decrypt}\ zyEnc\ \mathsf{as}\ \{M, A, B; znb\}_{K_B}\ \mathsf{in} \\
\quad (\nu\ K)\langle M, \{zna, K\}_{K_A}, \{znb, K\}_{K_B}\rangle.0)
\end{array}
$$

Table 3.8: A LySA process of the Otway-Rees protocol with only one initiator A and one responder B.

In the first line the keys K_A and K_B shared between A and the server S, and B and the server, respectively, are restricted. These restrictions model that the

keys are unknown to the outsiders. Technically, the scopes of K_A and K_B cover both A and B, which means, for example, K_A is available for B to use and K_B is available for A to use. This fact, although sounds confusing, actually does not matter: by inspecting the processes modelling both principals, it is easy to tell that A does not use the key K_B at all and similar for B. So this LySA process perfectly models that only A and S share the key K_A and only B and S share the key K_B.

The three lines after the restrictions model the action of the principal A, which 1) generates a new nonce N_A, 2) encrypts it together with the serial value M and the names of itself and the intended receiver using the shared key K_A, and 3) sends the value M, the source and destination address and the encryption onto the network. Note that the operator ! indicates that the principal A may execute these steps for any number of times.

The next four lines model the principal B, which 1) generates a nonce of it own N_B, 2) encrypts the nonce with other information using the shared key K_B, and 3) sends the encryptions and whatever it received from the network, i.e. ideally the encryption generated by A, onto the network.

The last four lines model the server S. The server 1) reads from network the encryptions generated by A and B, respectively, 2) decrypts them to get the nonces, N_A and N_B, 3) generates a fresh new session key K, 4) encrypts the new session key with the nonces, and 5) sends back to B.

3.3.1 Multiple Principals

Many protocols become vulnerable when more than one copies or sessions of the protocol are running simultaneously. This may give chances to an attack, i.e. messages from one session are used in another session running after it or in parallel.

Consider a scenario that there is a server for helping establish secure communications and a set of principals that each of them may act as either a initiator or a responder, as shown in Figure 3.1.

As far as the keys shared between principals and the server are concerned, this scenario can be further classified into two cases:

- each principal shares a pair of keys with the server, one is for the initiator's role and the other is for the responder's role. For example, a principal I

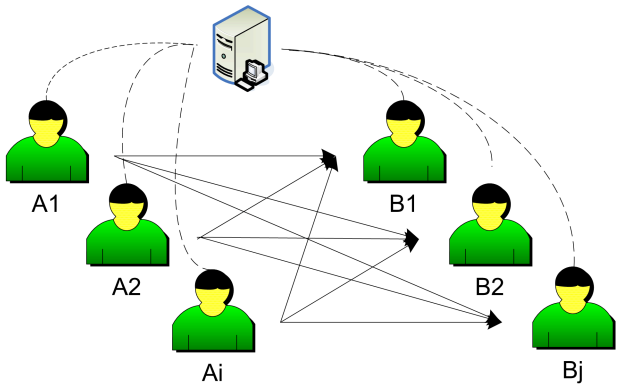


Figure 3.1: Multiple Principals Scenario

uses K_A to encrypt and decrypt when he acts as a initiator A, and uses K_B when acting as a responder.

- each principal share only one key with the server and use it for both initiating and responding purposes.

Consider the first case, where arbitrarily many principals A_i initiates sessions with arbitrarily many distinct responders B_j and each A_i uses the key K_{Ai} and each B_j uses the key K_{Bj}. The scenario may be modelled as in Table 3.9.

The meta level process is identical to the one listed in Table 3.8 except that indices have been added. The indexed parallel compositions and the indices have been added consistently such that names, variables used in a session between A_i and B_j are indexed ij.

We may also assume that although each principal may play the role of either the initiator or the responder, he uses the same key for the two roles. With this assumption about the key and others remaining the same, we can model the protocol as in Table 3.10. Note that, in the first line, the shared key KS_i is restricted for each principal, A_i and B_i, which is different from the restriction of K_{Ai} and K_{Bi} as in Table 3.9.

To summarize, in the section, we consider the scenario that a protocol is executed between two sets of distinct principals and each of them may choose to use different or the same key for the initiator or the responder roles. However there is a significant limitation of this scenario: the protocol establishment can only work in one direction in the sense that only principals A_i is allowed to initiate a session. In the next section, we shall consider a more general scenario.

let $X \subseteq S$ in $(\nu_{i\in X}\ K_{Ai})(\nu_{j\in X}\ K_{Bj})($

$|_{i\in X}|_{j\in X}$ $!(\nu\ N_{Aij})\langle M_{ij}, A_i, B_j, \{M_{ij}, A_i, B_j, N_{Aij}\}_{K_{Ai}}\rangle.$
$(M_{ij}; xEnc_{ij})$.decrypt $xEnc_{ij}$ as $\{N_{Aij}; xk_{ij}\}_{K_{Ai}}$ in
$(; xmEnc_{ij})$.decrypt $xmEnc_{ij}$ as $\{; xMsg_{ij}\}_{xk_{ij}}$ in 0

$||_{i\in X}|_{j\in X}$ $!(M_{ij}, A_i, B_j; yEnc_{ij}).$
$(\nu\ N_{Bij})\langle M_{ij}, A_i, B_j, yEnc_{ij}, \{M_{ij}, A_i, B_j, N_{Bij}\}_{K_{Bj}}\rangle.$
$(M_{ij}; yxEnc_{ij}, yzEnc_{ij})$.decrypt $yzEnc_{ij}$ as $\{N_{Bij}; yk_{ij}\}_{K_{Bj}}$ in
$\langle M_{ij}, yxEnc_{ij}\rangle.(\nu\ Msg_{ij})\langle\{Msg_{ij}\}_{yk_{ij}}\rangle.0$

$||_{i\in X}|_{j\in X}$ $!(M_{ij}, A_i, B_j; zxEnc_{ij}, zyEnc_{ij}).$
decrypt $zxEnc_{ij}$ as $\{M_{ij}, A_i, B_j; zna_{ij}\}_{K_{Ai}}$ in
decrypt $zyEnc_{ij}$ as $\{M_{ij}, A_i, B_j; znb_{ij}\}_{K_{Bj}}$ in
$(\nu\ K_{ij})\langle M_{ij}, \{zna_{ij}, K_{ij}\}_{K_{Ai}}, \{znb_{ij}, K_{ij}\}_{K_{Bj}}\rangle.0)$

Table 3.9: A LySA process of the Otway-Rees protocol where there are multiple principals and each of them uses different keys for initiator and responder roles

3.3.2 Bi-directional Key Establishment

The multiple principals scenario consists of two distinct sets of principals. Another, or more general, scenario is that it only consists of principals I_i such that each principal can act both as initiator and as responder of the protocols, e.g. each principal I_i initiates a session with every other principals I_j and meanwhile responds to the sessions initiated by each I_j, as shown in Figure 3.2. In such a scenario, the key establishment will be used in two directions.

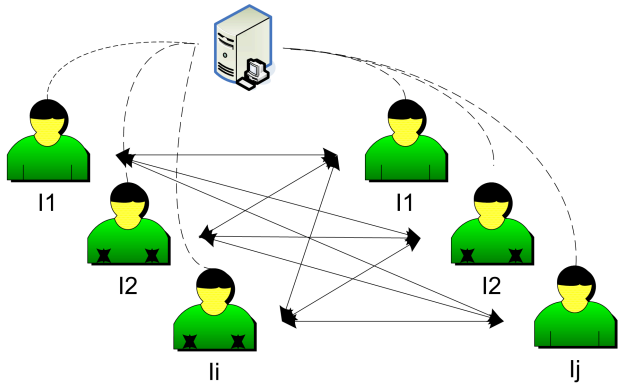


Figure 3.2: Bi-directional Key Establishment Scenario

$$
\begin{array}{l}
\mathsf{let}\ X \subseteq S\ \mathsf{in}\ (\nu_{i\in X}\ KS_i)(\\
\quad |_{i\in X}|_{j\in X} \quad !(\nu\ N_{Aij})\langle M_{ij}, A_i, B_j, \{M_{ij}, A_i, B_j, N_{Aij}\}_{KS_i}\rangle. \\
\qquad\qquad (M_{ij}; xEnc_{ij}).\mathsf{decrypt}\ xEnc_{ij}\ \mathsf{as}\ \{N_{Aij}; xk_{ij}\}_{KS_i}\ \mathsf{in} \\
\qquad\qquad (; xmEnc_{ij}).\mathsf{decrypt}\ xmEnc_{ij}\ \mathsf{as}\ \{; xMsg_{ij}\}_{xk_{ij}}\ \mathsf{in}\ 0 \\
\quad ||_{i\in X}|_{j\in X} \quad !(M_{ij}, A_i, B_j; yEnc_{ij}). \\
\qquad\qquad (\nu\ N_{Bij})\langle M_{ij}, A_i, B_j, yEnc_{ij}, \{M_{ij}, A_i, B_j, N_{Bij}\}_{KS_j}\rangle. \\
\qquad\qquad (M_{ij}; yxEnc_{ij}, yzEnc_{ij}).\mathsf{decrypt}\ yzEnc_{ij}\ \mathsf{as}\ \{N_{Bij}; yk_{ij}\}_{KS_j}\ \mathsf{in} \\
\qquad\qquad \langle M_{ij}, yxEnc_{ij}\rangle.(\nu\ Msg_{ij})\langle\{Msg_{ij}\}_{yk_{ij}}\rangle.0 \\
\quad ||_{i\in X}|_{j\in X} \quad !(M_{ij}, A_i, B_j; zxEnc_{ij}, zyEnc_{ij}). \\
\qquad\qquad \mathsf{decrypt}\ zxEnc_{ij}\ \mathsf{as}\ \{M_{ij}, A_i, B_j; zna_{ij}\}_{KS_i}\ \mathsf{in} \\
\qquad\qquad \mathsf{decrypt}\ zyEnc_{ij}\ \mathsf{as}\ \{M_{ij}, A_i, B_j; znb_{ij}\}_{KS_j}\ \mathsf{in} \\
\qquad\qquad (\nu\ K_{ij})\langle M_{ij}, \{zna_{ij}, K_{ij}\}_{KS_i}, \{znb_{ij}, K_{ij}\}_{KS_j}\rangle.0)
\end{array}
$$

Table 3.10: A LySA process of the Otway-Rees protocol where there are multiple principals and each of them uses the same key for initiator and responder roles

Similar to the multiple principals scenario, this scenario can be further classified into two cases concerning the keys used for each role.

- each principal I_i uses different keys to initiate and respond a session with each other principal I_j.
- each principal I_i uses the same key to initiate and respond a session with each other principal I_j

In the scenario that each principal I_i uses different keys to both initiate and respond a session, the Otway-Rees protocol can be modelled as a meta level process shown in Table 3.11. Shared keys K_{Ai} and K_{Bi} are restricted to each principal I_i. The principal I_i uses the key K_{Ai} to initiate a session and use the key K_{Bi} to respond a session. The server meditates communication between two arbitrary principals I_i and I_j. All the names and variables used in a session initiated by I_i to I_j have the index ij.

We may also choose to model the protocol running in a scenario that each principal I_i uses the same key KS_i to initiate and respond to a session with the principal I_j, as shown in Table 3.12.

$$
\begin{array}{l}
\mathsf{let}\ X \subseteq S\ \mathsf{in}\ (\nu_{i\in X}\ K_{Ai})(\nu_{j\in X}\ K_{Bj})(\\
\quad |_{i\in X}|_{j\in X} \quad !(\nu\ N_{Aij})\langle M_{ij}, I_i, I_j, \{M_{ij}, I_i, I_j, N_{Aij}\}_{K_{Ai}}\rangle. \\
\qquad\qquad (M_{ij}; xEnc_{ij}).\mathsf{decrypt}\ xEnc_{ij}\ \mathsf{as}\ \{N_{Aij}; xk_{ij}\}_{K_{Ai}}\ \mathsf{in} \\
\qquad\qquad (; xmEnc_{ij}).\mathsf{decrypt}\ xmEnc_{ij}\ \mathsf{as}\ \{; xMsg_{ij}\}_{xk_{ij}}\ \mathsf{in}\ 0 \\
\quad ||_{i\in X}|_{j\in X} \quad !(M_{ij}, I_i, I_j; yEnc_{ij}). \\
\qquad\qquad (\nu\ N_{Bij})\langle M_{ij}, I_i, I_j, yEnc_{ij}, \{M_{ij}, I_i, I_j, N_{Bij}\}_{K_{Bj}}\rangle. \\
\qquad\qquad (M_{ij}; yxEnc_{ij}, yzEnc_{ij}).\mathsf{decrypt}\ yzEnc_{ij}\ \mathsf{as}\ \{N_{Bij}; yk_{ij}\}_{K_{Bj}}\ \mathsf{in} \\
\qquad\qquad \langle M_{ij}, yxEnc_{ij}\rangle.(\nu\ Msg_{ij})\langle\{Msg_{ij}\}_{yk_{ij}}\rangle.0 \\
\quad ||_{i\in X}|_{j\in X} \quad !(M_{ij}, I_i, I_j; zxEnc_{ij}, zyEnc_{ij}). \\
\qquad\qquad \mathsf{decrypt}\ zxEnc_{ij}\ \mathsf{as}\ \{M_{ij}, I_i, I_j; zna_{ij}\}_{K_{Ai}}\ \mathsf{in} \\
\qquad\qquad \mathsf{decrypt}\ zyEnc_{ij}\ \mathsf{as}\ \{M_{ij}, I_i, I_j; znb_{ij}\}_{K_{Bj}}\ \mathsf{in} \\
\qquad\qquad (\nu\ K_{ij})\langle M_{ij}, \{zna_{ij}, K_{ij}\}_{K_{Ai}}, \{znb_{ij}, K_{ij}\}_{K_{Bj}}\rangle.0)
\end{array}
$$

Table 3.11: A LySA process of the Otway-Rees protocol where aribitrarily many principals I_i simultaneously act both as initiator and as responder and use different keys for the two roles

3.3.3 Insider Attacks

In modern distributed systems, it is normally the case that a group of principals is granted with some sorts of credentials in order to access a provided service. For example, in a university students use their student numbers and passwords to register/cancel courses, and in a large company employees access to the central server to retrieve clients information. However it is not plausible that all the users are trusted with all the sensitive information and there is no guarantee that all the principals with credentials always behave as expected. For example they may reveal their credentials, intentionally or accidently, to the attacker, who may then be able to launch an attack, or they may themselves use the credentials to compromise a secure communication between other honest principals, in this sense, these principals can be viewed as attackers. So in order to discuss insider attacks it is convenient to partition principals into *legitimate* ones, whose credentials are kept confidential and who will not launch attacks, and *illegitimate* ones, whose credential are revealed to the attacker or who may launch attacks.

In all the previous examples, we only model the *legitimate principals*, who only behave as they are suppose to do and we use the restriction operator to declare the scopes of the credentials of the legitimate principals, such as keys and nonces, i.e. they are initially protected from the attackers.

$$
\begin{array}{l}
\mathsf{let}\ X \subseteq S\ \mathsf{in}\ (\nu_{i\in X}\ KS_i)(\\
\quad |_{i\in X}|_{j\in X} \quad !(\nu\ N_{Aij})\langle M_{ij}, I_i, I_j, \{M_{ij}, I_i, I_j, N_{Aij}\}_{KS_i}\rangle. \\
\qquad\qquad (M_{ij}; xEnc_{ij}).\mathsf{decrypt}\ xEnc_{ij}\ \mathsf{as}\ \{N_{Aij}; xk_{ij}\}_{KS_i}\ \mathsf{in} \\
\qquad\qquad (; xmEnc_{ij}).\mathsf{decrypt}\ xmEnc_{ij}\ \mathsf{as}\ \{; xMsg_{ij}\}_{xk_{ij}}\ \mathsf{in}\ 0 \\
\quad ||_{i\in X}|_{j\in X} \quad !(M_{ij}, I_i, I_j; yEnc_{ij}). \\
\qquad\qquad (\nu\ N_{Bij})\langle M_{ij}, I_i, I_j, yEnc_{ij}, \{M_{ij}, I_i, I_j, N_{Bij}\}_{KS_j}\rangle. \\
\qquad\qquad (M_{ij}; yxEnc_{ij}, yzEnc_{ij}).\mathsf{decrypt}\ yzEnc_{ij}\ \mathsf{as}\ \{N_{Bj}; yk_{ij}\}_{KS_j}\ \mathsf{in} \\
\qquad\qquad \langle M_{ij}, yxEnc_{ij}\rangle.(\nu\ Msg_{ij})\langle\{Msg_{ij}\}_{yk_{ij}}\rangle.0 \\
\quad ||_{i\in X}|_{j\in X} \quad !(M_{ij}, I_i, I_j; zxEnc_{ij}, zyEnc_{ij}). \\
\qquad\qquad \mathsf{decrypt}\ zxEnc_{ij}\ \mathsf{as}\ \{M_{ij}, I_i, I_j; zna_{ij}\}_{KS_i}\ \mathsf{in} \\
\qquad\qquad \mathsf{decrypt}\ zyEnc_{ij}\ \mathsf{as}\ \{M_{ij}, I_i, I_j; znb_{ij}\}_{KS_j}\ \mathsf{in} \\
\qquad\qquad (\nu\ K_{ij})\langle M_{ij}, \{zna_{ij}, K_{ij}\}_{KS_i}, \{znb_{ij}, K_{ij}\}_{KS_j}\rangle.0)
\end{array}
$$

Table 3.12: A LySA process of the Otway-Rees protocl where aribitraryly many principals I_i simultaneously act both as initiator and as responder and use the same key for the two roles

In this section, we shall consider the scenario where both legitimate principals and illegitimate principals exist and they are all allowed to initiate and respond to a session with another principals, either legitimate or illegitimate. Furthermore, the server is allowed to meditate communications between both legitimate and illegitimate principals.

Consider the process in Table 3.12. To take the illegitimate principals into account, the Otway-Rees protocol can be modelled as in Table 3.13.

To model both legitimate and illegitimate principals, the set of principals I_i is partitioned into two sets: the legitimate principals have an index $i \in X \subseteq S$ and the illegitimate principals take the index $i \in \{0\}$. In the first line, the keys of illegitimate principals, e.g. KS_0, are not restricted and therefore are known to the attacker. This enables the attacker to act as illegitimate principals that may attack the legitimate part of the protocol. The parallel composition $|_{j\in Y}$ in the initiator's process models that each I_i may initiate a session with an illegitimate principal. Similarly, $|_{i\in Y}$ in the responder's process allows each principal to response to a session initiated by an illegitimate principal, and $|_{i\in Y}|_{j\in Y}$ allows the server to communicate with either legitimate or illegitimate principals.

$$
\begin{array}{l}
\textsf{let } X \subseteq S \textsf{ in } (\nu_{i\in X}\ KS_i)(\\
\textsf{let } Y \subseteq X \cup \{0\} \textsf{ in} \\
\quad |_{i\in X}|_{j\in Y} \quad !(\nu\ N_{Aij})\langle M_{ij}, I_i, I_j, \{M_{ij}, I_i, I_j, N_{Aij}\}_{KS_i}\rangle. \\
\qquad\qquad (M_{ij}; xEnc_{ij}).\textsf{decrypt } xEnc_{ij} \textsf{ as } \{N_{Aij}; xk_{ij}\}_{KS_i} \textsf{ in} \\
\qquad\qquad (; xmEnc_{ij}).\textsf{decrypt } xmEnc_{ij} \textsf{ as } \{; xMsg_{ij}\}_{xk_{ij}} \textsf{ in } 0 \\
\quad ||_{i\in Y}|_{j\in X} \quad !(M_{ij}, I_i, I_j; yEnc_{ij}). \\
\qquad\qquad (\nu\ N_{Bij})\langle M_{ij}, I_i, I_j, yEnc_{ij}, \{M_{ij}, I_i, I_j, N_{Bij}\}_{KS_j}\rangle. \\
\qquad\qquad (M_{ij}; yxEnc_{ij}, yzEnc_{ij}).\textsf{decrypt } yzEnc_{ij} \textsf{ as } \{N_{Bij}; yk_{ij}\}_{KS_j} \textsf{ in} \\
\qquad\qquad \langle M_{ij}, yxEnc_{ij}\rangle.(\nu\ Msg_{ij})\langle\{Msg_{ij}\}_{yk_{ij}}\rangle.0 \\
\quad ||_{i\in Y}|_{j\in Y} \quad !(M_{ij}, I_i, I_j; zxEnc_{ij}, zyEnc_{ij}). \\
\qquad\qquad \textsf{decrypt } zxEnc_{ij} \textsf{ as } \{M_{ij}, I_i, I_j; zna_{ij}\}_{KS_i} \textsf{ in} \\
\qquad\qquad \textsf{decrypt } zyEnc_{ij} \textsf{ as } \{M_{ij}, I_i, I_j; znb_{ij}\}_{KS_j} \textsf{ in} \\
\qquad\qquad (\nu\ K_{ij})\langle M_{ij}, \{zna_{ij}, K_{ij}\}_{KS_i}, \{znb_{ij}, K_{ij}\}_{KS_j}\rangle.0)
\end{array}
$$

Table 3.13: A LySA process of the Otway-Rees protocol with both legitimate and illegitimate principals

3.4 Why Process Calculus

In the protocol analysis literature, there has long been many formalisms for formally describing security protocols. The main reason to use formal description techniques is that they give rise to both analyses at the specification level and the tools at the implementation level. Perhaps a very difficult part of analysing protocols is in finding a proper formalism to use, as there are many of them to choose from.

One of them is Petri nets, which is a graphical and mathematical modelling tool that is applicable for describing and studying security protocols. Petri nets and its extensions, e.g. colored petri nets [53] and numerical petri nets [80], have been proved to be successful in modelling protocols. Some of the research and applications include [12, 13, 32, 31, 54, 60]. However, careful attention must be paid to tradeoff between the level of abstraction and the capability of the analyses. In fact, a major weak point of petri net is the complexity problem, i.e. petri net based models may become very large for the analysis of even a modest-size protocol.

Another approach that seems promising for modelling security protocols is modal logic. Within this approach, the protocol, and necessary assumptions and the goals of the protocols are formulated in formal logic, which consists of

primitive propositions, represented by propositional letters, and propositional connectives, e.g. *or*, *and*, *not* and *implication*. The best known one of the logics was developed by Burrows, Abadi and Needham [23], which is usually called BAN logic. Although it is very useful, its relation to implementation may be quite tenuous and subtle [3].

Process calculus is yet another way to model security protocols. It is a model based on modelling processes and events. Process calculus is a small, programming-like language. It is directly executable and it has a precise semantics to describe how processes are evolved.

In this section, we reviewed a process calculus, LySA, which incorporates cryptographic primitives, and was designed especially for modelling security protocols. We showed how to encode in LySA various scenarios in which a protocol is deployed. Encoding protocols in LySA gives a formal and precise way of describing protocol executions and therefore facilitates analysing. In the next chapter, we shall present a control flow analysis which precisely collects the possible behaviour of a given protocol.

Chapter 4

Control Flow Analysis

Control flow analysis comes from the field known as static analysis. Static analysis examines a piece of code of a program statically, without attempting to execute it. However, not all the non-trivial questions one may ask about a program are computable given finite amount of resources, e.g. time and memory, as stated by Rice's theorem [75]. In other words, static analysis problems are *undecidable* in the worse case. A tradeoff one has to make between preciseness and decidability is then to force static analysis to give approximate answers about a property of a piece of code. It means that the answer may include false positives, e.g. a bug that the program doesn't contain. However, this is the price one has to pay for gaining the decidability.

Originally, static analysis was developed for generating codes and optimising compilers [62, 24]. Nevertheless, many static analysis approaches have recently been directed to apply to security. Encouraging results have been obtained by the use of those approaches, such as type systems [48, 56] and control flow analysis. These techniques predict safe and computable approximations to the set of values or behaviours arising dynamically during run-time. For example, the control flow analysis for LySa process given in [15, 16, 20] computes an approximation of the run-time behaviour of a process. The analysis has been applied to prove the authentication property of protocols.

Example 4.1 *Consider the simple protocol from Example 2.1, which has a*

LySA encoding as the following,

$$
\begin{array}{l}
(\nu\ K)(\\
\quad (\nu\ CCI)\langle A, B, \{CCI\}_K\rangle.0 \\
\quad | \\
\quad (A, B; y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI\}_K\ \mathsf{in}\ 0)
\end{array}
$$

In this protocol, it is of interest to investigate that whether the credit card information CCI is received by B correctly. To do this, the control flow analysis should be able to determine which value may be bound to the variable $yCCI$. However, the value is not bound to $yCCI$ until the decryption of y is successful. To trace back the value being bound to y, the analysis also has to collect the messages which flow over the network.

In this case, we are expecting the analysis to figure out that:

- *the value CCI is bound to the variable $yCCI$, and*
- *the message $\langle A, B, \{CCI\}_K\rangle$ is sent over the network*

Before going into the detail of the control flow analysis, we shall briefly introduce the concept of flow logic, in which the control flow analysis is formalised.

4.1 Flow Logic

Flow Logic is relatively new concept in the program analysis field. It was introduced in the late 90's by Nielson, Nielson [74, 72, 73] and Hankin [71], and has been used for analysis of a wide variety of languages and process calculi, e.g. the $\pi-$calculus [66], Spi-calculus [3], $\lambda-$calculus [9], the ambient calculus [25], imperative objects, Concurrent ML, object-oriented constructs, and protocol narrations.

Flow logic is a formalism based on logical systems for specifying static analysis. It separates the *specification* of an analysis and the actual *computation* of an analysis result. It therefore, on one hand, allows one to focus the effort on designing and specifying what it means for an analysis estimate to be acceptable for a program without being bothered by implementation considerations, and, on the other hand, benefits the implementation of the analysis by allowing one to freely chooses existing suitable tools to do the computations. In this sense,

the Flow Logic framework can be seen as a "specification approach" to static analysis, rather than an "implementation approach".

A control flow analysis of a process P works by collecting information about some aspects of the behaviour of P. This information is stored in some data structures, say $\mathcal{A}_1, \ldots, \mathcal{A}_k$, which are called *analysis components*, and each analysis component holds one aspect of the process behaviour. As mentioned before, flow logic specification focuses on the relationship between an analysis estimate and the process to be analysed. Formally, this relationship is captured by an acceptability judgement, written as

$$\mathcal{A}_1, \ldots, \mathcal{A}_k \models P$$

which holds precisely when $\mathcal{A}_1, \ldots, \mathcal{A}_k$ are descriptions of the behaviour of the process P.

4.1.1 Verbose and Succinct Flow Logics

A flow logic specification can be classified by the criteria

- *succinct* versus *verbose*

depending on the scopes of the analysis components.

The most common format of a flow logic specification is the *verbose* one, which has a form as,

$$\mathcal{A}_1, \ldots, \mathcal{A}_k \models P \quad \text{iff} \quad \textit{a logic formula } \mathcal{F} \textit{ holds}$$

and means that in order for $\mathcal{A}_1, \ldots, \mathcal{A}_k$ to be an estimate of the process P, the logic formula $\mathcal{F}$ has to hold. In this format, an analysis is specified by structurally defining a logic formula $\mathcal{F}$ for each syntactic construct of the process P. In general, the formula $\mathcal{F}$ can be an arbitrary formula in logical form and may even be recursive, i.e. it may contain $\mathcal{A}_1, \ldots, \mathcal{A}_k \models P'$ for an arbitrary process P'.

Example 4.2 *The rule*

$$\mathcal{A}_1, \ldots, \mathcal{A}_k \models P_1 \mid P_2 \;\; \textit{iff} \;\; \mathcal{A}_1, \ldots, \mathcal{A}_k \models P_1 \;\wedge\; \mathcal{A}_1, \ldots, \mathcal{A}_k \models P_2$$

is a verbose flow logic specification for analysing parallel composition of processes. It defines that in order for $\mathcal{A}_1, \ldots, \mathcal{A}_k$ to be an estimate of the process $P_1 \mid P_2$, it has to be estimates for both P_1 and P_2.

In the *verbose* form, all the analysis components, i.e. $\mathcal{A}_1, \ldots, \mathcal{A}_k$ are implicitly assumed to be *global*, i.e. their scopes are up to all the analysis rules.

A *succinct* flow logic specification, on the contrary, records information about a process locally, by the rules of the form

$$\mathcal{A}_1, \ldots, \mathcal{A}_k \models P : \mathcal{A}' \quad \text{iff} \quad \textit{a logic formula } \mathcal{F} \textit{ holds}$$

where $\mathcal{A}'$ is another analysis component holding the information only about the process P. This succinct component is implicitly assumed to be *local* to this formula and therefore is not known anywhere else in the entire analysis.

Example 4.3 *The rules*

$$\begin{array}{l} \mathcal{A}_1, \ldots, \mathcal{A}_k \models n_1 : \mathcal{A}' \;\; \textit{iff} \;\; n_1 \in \mathcal{A}' \;\; \textit{and} \\ \mathcal{A}_1, \ldots, \mathcal{A}_k \models n_2 : \mathcal{A}' \;\; \textit{iff} \;\; n_2 \in \mathcal{A}' \end{array}$$

are succinct flow logic specifications. The analysis component $\mathcal{A}'$ in the two rules may not have the same content because of the different scopes.

The next section will present a control flow analysis of LySA in the style of flow logic.

4.2 A Control Flow Analysis of LySA

The LySA calculus is especially designed to model security protocols involving a number of principals, where each of them executes a sequence of actions, synchronised by communications. Because of the inter-actions, in most of the cases, it is impossible to predict the exact behaviour of each principal. In this section, we present a control flow analysis aiming at collecting the central aspect of the information of a protocol of interest. This is done by over-approximating the protocol behaviour along all the execution paths, as shown in Figure 4.1.

4.2.1 Domain of the Analysis

The control flow analysis describes a protocol behaviour by collecting all the communications that a process may participate in. This information, i.e. the tuples of values that may be communicated over the network, is recorded in an analysis component κ. As said before, successful communications involve pattern matching and variable binding, i.e. binding values to variables. It

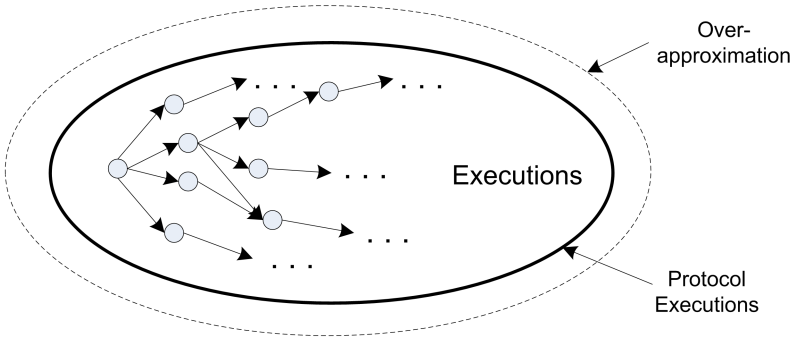


Figure 4.1: The relationship between the protocol behaviour and analysis

is handy and convenient for the analysis to collect this information. For this purpose, another analysis component ρ is introduced to record the values that each variable may be bound to. Thus the analysis components in the control flow analysis of LySA are κ and ρ, corresponding to the abstract representations $\mathcal{A}_1$ and $\mathcal{A}_2$ in Section 4.1.

4.2.1.1 Name Space

Both the analysis components κ and ρ have to do with recording values $V \in Val$ in some format. However, a LySA process may generate infinitely many values during an execution because of the restriction and replication constructs, which means that the analysis components have to be able to record infinitely many names.

For keeping the analysis component finite, a solution is to partition all the names used by a process into finitely many equivalence classes and use the names of the equivalence classes instead of the actual names. This partition works in a way that names from the same equivalence class are assigned a common *canonical name* and consequently there are only finitely many canonical names in any execution of a given process. This is enforced by assigning the same canonical name to every name generated by the same restriction. The canonical name for a name n is written as $\lfloor n \rfloor$ and for a value V written as $\lfloor V \rfloor$. The analysis component will then use canonical values taken from the set $\lfloor Val \rfloor$ ranged over by U.

Example 4.4 *An example process, that may generate infinitely many names,*

is $!(\nu\ n)P$*, which has one of the computational sequences as*

$$!(\nu\ n)P \rightarrow (\nu\ n')P' \mid !(\nu\ n)P \rightarrow (\nu\ n')P' \mid (\nu\ n'')P'' \mid !(\nu\ n)P \rightarrow \ldots$$

Furthermore, the names n*,* n' *and* n'' *are generated by the same restriction and hence have the same canonical name, i.e.* $\lfloor n \rfloor = \lfloor n' \rfloor = \lfloor n'' \rfloor$

4.2.1.2 Analysis Components

The analysis components of the control flow analysis of LySA are κ and ρ.

The κ component records the tuples of canonical values corresponding to the tuples of values that may be communicated over network during a process execution. Formally, we have

$$\kappa \in \mathcal{P}(\lfloor Val \rfloor^*)$$

The ρ component records a set of canonical values, for each variable, corresponding to the set of values that each variable may become bound to during a process execution. This can be formally written as

$$\rho : \lfloor Var \rfloor \rightarrow \mathcal{P}(\lfloor Val \rfloor)$$

4.2.2 Definition of the Analysis

The result of analysing a process P is a pair (ρ, κ). The first component, ρ, is an abstract environment which gives information about the set of canonical values to which names can be bound; the second component, κ, is an abstract network environment which gives information about tuples of values that can flow over the network. To be more precise, we have,

$$\rho, \kappa \models P$$

expressing that ρ and κ are valid analysis estimates of the process P. The judgement is defined by the axioms and rules in the lower part of Table 4.1 and is explained later.

The definition of the analysis for expressions makes use of an auxiliary predicate ϑ and takes the form,

$$\rho \models E : \vartheta$$

for analysing expressions. This judgement is in a succinct form of flow logic and expresses that $\vartheta \subseteq \mathcal{P}(\lfloor Val \rfloor)$ is an acceptable estimate of the set of canonical values that E may evaluate to given the environment ρ. It is defined inductively in the structure of the expressions in the upper part of Table 4.1.

The rules (AN), (ANp), (ANm) express that names are evaluated to their canonical representatives, by requiring that the corresponding canonical names are in ϑ.

The rule (AVar) says that a variable is evaluated to the values described by ρ for the corresponding canonical variables.

The rules (AEnc) and (AAEnc) evaluate $k-$tuple symmetric and asymmetric encryptions, respectively. They are defined in a similar way. The rules first recursively evaluate all the sub-expressions in the encryption, then compute all the $k-$ary encrypted values that can be formed by combining the values that the sub-expressions may be evaluated to, and finally require that those values are in ϑ. Essentially, these two rules replace all the variables in the encryption by their canonical values recorded in ρ.

The rest of the rules are defined for analysing processes.

The rule (AOut) analyses the output construct and does two things: first, all the expressions are evaluated and then it is required that all the combinations of the values found by this evaluation are recorded in κ. Finally, the continuation process must be analysed.

The rule (AInp) analyses the input construct. It incorporates pattern matching, which is dealt with by first evaluating all the first j expressions in the input to be the sets ϑ_i for $i = 1, \ldots, j$. Next, if any of the sequences of length k in κ is such that the first j values component-wise are included in ϑ_i then the match is concluded to be successful. In this case, the remaining values of the k-tuple must be recorded in ρ as possible bindings of the variables. Finally, the continuation process must be analysed.

The rules (ADec) and (AADec) analyse symmetric and asymmetric decryption, respectively. They handle the matching similarly to the rule for *input*, (AInp). The only difference is that here the matching is performed against $j + 1$ components including the key. In case of symmetric decryption, the key used for encryption and the one used for decryption have to match with each other. In the case of asymmetric decryption, the two keys, for encryption and decryption, have to form a key pair, m^+ and m^-. After the successful matching, values are bound to the corresponding variables.

(AN)	$\rho \models n : \vartheta$	iff $\lfloor n \rfloor \in \vartheta$
(ANp)	$\rho \models m^+ : \vartheta$	iff $\lfloor m^+ \rfloor \in \vartheta$
(ANm)	$\rho \models m^- : \vartheta$	iff $\lfloor m^- \rfloor \in \vartheta$
(AVar)	$\rho \models x : \vartheta$	iff $\rho(\lfloor x \rfloor) \subseteq \vartheta$
(AEnc)	$\rho \models \{E_1, \ldots, E_k\}_{E_0} : \vartheta$	
		iff $\wedge_{i=0}^{k}\ \rho \models E_i : \vartheta_i\ \wedge$
		$\quad \forall U_0, \ldots, U_k : \wedge_{i=0}^{k}\ U_i \in \vartheta_i \Rightarrow$
		$\quad\quad \{U_1, \ldots, U_k\}_{U_0} \in \vartheta$
(AAEnc)	$\rho \models \{\!|E_1, \ldots, E_k|\!\}_{E_0} : \vartheta$	
		iff $\wedge_{i=0}^{k}\ \rho \models E_i : \vartheta_i\ \wedge$
		$\quad \forall U_0, \ldots, U_k : \wedge_{i=0}^{k}\ U_i \in \vartheta_i \Rightarrow$
		$\quad\quad \{\!|U_1, \ldots, U_k|\!\}_{U_0} \in \vartheta$

(AOut)	$\rho, \kappa \models \langle E_1, \ldots, E_k \rangle.P$	
		iff $\wedge_{i=1}^{k}\ \rho \models E_i : \vartheta_i\ \wedge$
		$\quad \forall U_1, \ldots, U_k : \wedge_{i=1}^{k} U_i \in \vartheta_i \Rightarrow$
		$\quad\quad (\langle U_1, \ldots, U_k \rangle \in \kappa\ \wedge\ \rho, \kappa \models P)$
(AInp)	$\rho, \kappa \models (E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P$	
		iff $\wedge_{i=1}^{j}\ \rho \models E_i : \vartheta_1\ \wedge$
		$\quad \forall \langle U_1, \ldots, U_k \rangle \in \kappa : \wedge_{i=1}^{j}\ U_i \in \vartheta_i \Rightarrow$
		$\quad\quad (\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor)\ \wedge \rho, \kappa \models P)$
(ADec)	$\rho, \kappa \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}_{E_0}\ \mathsf{in}\ P$	
		iff $\rho \models E : \vartheta\ \wedge\ \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i \wedge$
		$\quad \forall \{U_1, \ldots, U_k\}_{U_0} \in \vartheta\ \wedge$
		$\quad\quad \wedge_{i=0}^{j}\ U_i \in \vartheta_i \Rightarrow$
		$\quad\quad\quad (\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor)\ \wedge \rho, \kappa \models P)$
(AADec)	$\rho, \kappa \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j; x_{j+1}, \ldots, x_k|\!\}_{E_0}\ \mathsf{in}\ P$	
		iff $\rho \models E : \vartheta\ \wedge\ \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i \wedge$
		$\quad \forall \{\!|U_1, \ldots, U_k|\!\}_{U_0} \in \vartheta : \forall U_0' \in \vartheta_0 :$
		$\quad \forall (m^+, m^-) : \{U_0, U_0'\} = \{\lfloor m^+ \rfloor, \lfloor m^- \rfloor\} \wedge$
		$\quad \wedge_{i=1}^{j}\ U_i \in \vartheta_i \Rightarrow$
		$\quad\quad (\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor)\ \wedge \rho, \kappa \models P)$
(ANew)	$\rho, \kappa \models (\nu\ n)P$	iff $\rho, \kappa \models P$
(AANew)	$\rho, \kappa \models (\nu_{\pm}\ m)P$	iff $\rho, \kappa \models P$
(APar)	$\rho, \kappa \models P_1 \vert P_2$	iff $\rho, \kappa \models P_1\ \wedge\ \rho, \kappa \models P_2$
(ARep)	$\rho, \kappa \models !P$	iff $\rho, \kappa \models P$
(ANil)	$\rho, \kappa \models 0$	iff true

Table 4.1: Analysis of terms; $\rho \models E : \vartheta$, and processes: $\rho, \kappa \models P$

The rules (ANew) and (AANew) require that the sub-process P has to be analysed.

The rule (APar) requires that the two sub-processes P_1 and P_2 are analysed.

The rule (ARep) requires that the sub-process is analysed.

The rule (ANil) trivially holds and does not put any requirement on the analysis components.

Example 4.5 *Consider the process from Example 4.1. We shall apply the control flow analysis to the process to illustrate how the analysis works and verify whether the analysis gives the expected results.*

$$
\begin{array}{l}
(\nu\ K)(\\
\quad (\nu\ CCI)\langle A, B, \{CCI\}_K\rangle.0 \\
\quad | \\
\quad (A, B; y).\mathit{decrypt}\ y\ \mathit{as}\ \{; yCCI\}_K\ \mathit{in}\ 0)
\end{array}
$$

Semantically, the two parallel processes can make a communication, where the message $\langle A, B, \{CCI\}_K\rangle$ is sent onto the network, which results in binding the value $\{CCI\}_K$ to y and the further decryption of y then binds CCI to the variable $yCCI$. Finally the process evolves into $0 \mid 0$.

The analysis rules from Table 4.1, that are applicable to this example, are:

$$\rho, \kappa \models (\nu\ CCI)(\langle A, B, \{CCI\}_K\rangle.0 \mid (A, B; y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI\}_K\ \mathsf{in}\ 0)$$

$$\text{iff} \quad \rho, \kappa \models \langle A, B, \{CCI\}_K\rangle.0 \mid (A, B; y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI\}_K\ \mathsf{in}\ 0 \qquad \text{(ANew)}$$

$$\begin{array}{ll}
\text{iff} & \rho, \kappa \models \langle A, B, \{CCI\}_K\rangle.0 \wedge \\
& \rho, \kappa \models (A, B; y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI\}_K\ \mathsf{in}\ 0 \qquad \text{(APar)}
\end{array}$$

$$\begin{array}{ll}
\text{iff} & \rho \models A : \vartheta_1\ \wedge\ \rho \models B : \vartheta_2\ \wedge\ \rho \models \{CCI\}_K : \vartheta_3\ \wedge \\
& \forall U_1, U_2, U_3 : \wedge_{i=1}^{3} U_i \in \vartheta_i \Rightarrow \langle U_1, U_2, U_3\rangle \in \kappa\ \wedge\ \rho, \kappa \models 0\ \wedge \qquad \text{(AOut)} \\
& \rho \models A : \vartheta_1\ \wedge\ \rho \models B : \vartheta_2\ \wedge\ \forall \langle U_1, U_2, U_3\rangle \in \kappa : \wedge_{i=1}^{2}\ U_i \in \vartheta_i \Rightarrow \\
& U_3 \in \rho(\lfloor y \rfloor)\ \wedge \rho, \kappa \models \mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI\}_K\ \mathsf{in}\ 0 \qquad \text{(AInp)}
\end{array}$$

$$\begin{array}{ll}
\text{iff} & \rho \models y : \vartheta\ \wedge\ \rho \models K : \vartheta_0\ \wedge\ \forall \{U_1\}_{U_0} \in \vartheta\ \wedge\ U_0 \in \vartheta_0 \Rightarrow \\
& U_1 \in \rho(\lfloor yCCI \rfloor)\ \wedge \rho, \kappa \models 0 \qquad \text{(ADec)}
\end{array}$$

The least solution to these constraints is,

$$\{\{\lfloor CCI \rfloor\}_{\lfloor K \rfloor}\} \subseteq \rho(\lfloor y \rfloor)$$
$$\{\lfloor CCI \rfloor\} \subseteq \rho(\lfloor yCCI \rfloor)$$
$$\{\langle \lfloor A \rfloor, \lfloor B \rfloor, \{\lfloor CCI \rfloor\}_{\lfloor K \rfloor} \rangle\} \subseteq \kappa$$

4.2.3 The Meta Level Analysis

An object level analysis, presented above, gives an account of the dynamic behaviour of an object level process. A meta level M represents a set of object level processes that it may be instantiated to. The meta level analysis will therefore give an account of the dynamic behaviour of those object level processes.

The meta level process is an extension of the object level process with additional sequence of indices, $i_1, \ldots, i_k$, to names and variables. Consequently, the meta level analysis relies on the object level analysis and extends it to account for the instantiation of the meta level constructs.

The meta level analysis is defined in Table 4.2, which is of the form

$$\rho, \kappa \models_\Gamma M$$

where $\Gamma : SetID \cup \mathcal{P}(Index_{fin}) \to \mathcal{P}(Index_{fin})$ is a mapping from set identifiers to finite sets.

Because the let- construct, e.g. $\mathsf{let} X \subseteq S \mathsf{in} MP$, may define an infinite indexing set S, a meta level process may instantiate to infinitely many object level processes. This problem is solved by partitioning each index set S into finitely many equivalent classes, where each class contains indexes having the same canonical representatives.

The rule (MLet) analyses the let-construct. It updates the environment Γ with the mapping $X \mapsto S'$, where S' is required to be a finite set and has the same canonical names as the set S.

The rule (MIPar) requires that the analysis holds for all the processes M where the index i is substituted by all the elements in $\Gamma(S)$.

The rules (MINew) and (MIANew) simply ignore the restriction operators.

(MLet)	$\rho, \kappa \models_\Gamma \mathsf{let}\ X \subseteq S\ \mathsf{in}\ M$	iff	$\rho, \kappa \models_{\Gamma[X \mapsto S']} M$ where $S' \subseteq_{fin} \Gamma(S)$ and $\lfloor S' \rfloor = \lfloor \Gamma(S) \rfloor$
(MIPar)	$\rho, \kappa \models_\Gamma \mid_{i \in S} M$	iff	$\wedge_{a \in \Gamma(S)} \rho, \kappa \models_\Gamma M[i \mapsto a]$
(MINew)	$\rho, \kappa \models_\Gamma (\nu_{\bar{i} \in \bar{S}}\ n_{\bar{a}\bar{i}})$	iff	$\rho, \kappa \models_\Gamma M$
(MIANew)	$\rho, \kappa \models_\Gamma (\nu_{\pm\bar{i} \in \bar{S}}\ m_{\bar{a}\bar{i}})$	iff	$\rho, \kappa \models_\Gamma M$
(MOut)	$\rho, \kappa \models_\Gamma \langle ME_1, \ldots, ME_k \rangle.M$	iff	$\wedge_{i=1}^k \rho \models ME_i : \vartheta_i \wedge$ $\forall U_1, \ldots, U_k : \wedge_{i=1}^k U_i \in \vartheta_i \Rightarrow$ $\langle U_1, \ldots, U_k \rangle \in \kappa \ \wedge\ \rho, \kappa \models_\Gamma M$
(MN)	$\rho \models n_{\bar{i}} : \vartheta$	iff	$\lfloor n_{\bar{i}} \rfloor \in \vartheta$
(MVar)	$\rho \models x_{\bar{i}} : \vartheta$	iff	$\rho(\lfloor x_{\bar{i}} \rfloor) \subseteq \vartheta$

Table 4.2: Analysis of meta level processes. The remaining cases of the analysis where the meta level and the object level overlap are as in Table 4.1 but using meta level expressions and meta level variables instead of the object level ones.

The rest of the rules are similar to the ones for analysing object level, listed in Table 4.1 except that they range over indexed names and variables.

4.3 The Attacker

Communications between principals over a computer network are vulnerable to mischievous behaviour of other parties who have access to the network. This is the typical scenario for protocol executions. Hence the attacker's involvement has to be taken into account when analysing cryptographic protocols. In this section, we shall briefly recall how the control flow analysis is used to analyse protocols running in a malicious environment.

In LySA, the protocol and the attacker are modelled, formally, as two parallel processes, $P_{sys} \mid P_\bullet$, P_{sys} represents the protocol process and $P_\bullet$ is some arbi-

trary attacker. The attacker considered here is the one proposed by Needham and Schroeder [70] and later one formalised by Dolev and Yao [33], which is often referred to as the Dolev-Yao attacker (or Dolev-Yao threat model). The Dolev-Yao attacker is an active attacker and assumed to have the overall control of the network, over which principals exchange messages. Therefore he has access to messages transmitted over the network and is able to perform all kinds of operations of the messages within the permission of the semantics of the LySA, e.g. he is not only able to eavesdrop or replay messages sending over the network but also to encrypt, decrypt or generate messages providing that the necessary information is within his knowledge. On the other hand, however, the secret messages and keys, e.g. $(\nu\ Key_{ab})$, are restricted to their scope in P_{sys} and thus not immediately accessible to the attacker.

The attacker's process is constructed in the way that all the attackers are characterised and as the following.

Not to lose generality, some basic information of the protocol process, i.e. type of the process, has to be abstracted out. A process P_{sys} has the type whenever: (1) it is close, (2) all the free names of P_{sys} are in $\mathcal{N}_{\mathsf{f}}$, (3) all the arities of the sent out or received messages are in $\mathcal{A}_{\kappa}$ and (4) all the arities of the encrypted or decrypted messages are in $\mathcal{A}_{\mathsf{Enc}}$. Obviously, $\mathcal{N}_{\mathsf{f}}$, $\mathcal{A}_{\kappa}$ and $\mathcal{A}_{\mathsf{Enc}}$ are all finite and can be calculated out by inspecting the process P_{sys}.

One concern regarding the attacker process is the names and variables it used, which have to be distinct from the ones used by P_{sys}. Let all the names used by P_{sys} to be in a finite set $\mathcal{N}_{\mathsf{c}}$ and all the variables in a finite set $\mathcal{X}_{\mathsf{c}}$, a new name, $n_{\bullet}$, is postulated , where $n_{\bullet}$ is not in $\mathcal{N}_{\mathsf{c}}$, and a new variable $z_{\bullet}$ not in $\mathcal{X}_{\mathsf{c}}$. This means no overlapping between the names and variables used by the legitimate principals and the ones used by the attacker.

In order to control the number of names and variables used by the attacker, a semantically equivalent process, $\overline{P'}$, was constructed , for a process $P_{\bullet}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_{\kappa}, \mathcal{A}_{\mathsf{Enc}})$, as follows: 1) all restrictions $(\nu\ n)P$ are $\alpha-$converted into restrictions $(\nu\ n')P$ where n' has the canonical representative $n_{\bullet}$, 2) all the occurrences of variables x_i in $(E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P$ and
decrypt E *as* $\{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}$ *in* P are $\alpha-$converted to use variables x'_i with canonical representative $z_{\bullet}$. Therefore $\overline{P_{\bullet}}$ only has finitely many canonical names and variables.

A formula $\mathcal{F}^{DY}_{RM}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_{\kappa}, \mathcal{A}_{\mathsf{Enc}})$ was defined as the conjunction of the five components in Table 4.3, where each describes an ability of the attacker. Furthermore, it was proved that the formula $\mathcal{F}^{DY}_{RM}$ is capable of characterising the potential effect of all attackers $P_{\bullet}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_{\kappa}, \mathcal{A}_{\mathsf{Enc}})$.

(1) $\wedge_{k\in\mathcal{A}_\kappa}\ \forall\,\langle V_1,\ldots,V_k\rangle\in\kappa:\ \wedge_{i=1}^{k}V_i\in\rho(z_\bullet)$
the attacker may learn by eavesdropping

(2) $\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{V_1,\ldots,V_k\}_{V_0}\in\rho(z_\bullet):V_0\in\rho(z_\bullet)\Rightarrow\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)$
$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{\!|V_1,\ldots,V_k|\!\}_{m^+}\in\rho(z_\bullet):m^-\in\rho(z_\bullet)\Rightarrow\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)$
$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{\!|V_1,\ldots,V_k|\!\}_{m^-}\in\rho(z_\bullet):m^+\in\rho(z_\bullet)\Rightarrow\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)$
the attacker may learn by decrypting messages with keys already known

(3) $\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\,V_0,\ldots,V_k:\wedge_{i=0}^{k}V_i\in\rho(z_\bullet)\Rightarrow\{V_1,\ldots,V_k\}_{V_0}\in\rho(z_\bullet)$
$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\,m^+,V_1\ldots,V_k:m^+\in\rho(z_\bullet)\wedge\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\Rightarrow$
$\{\!|V_1,\ldots,V_k|\!\}_{m^+}\in\rho(z_\bullet)$
$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\,m^-,V_1\ldots,V_k:m^-\in\rho(z_\bullet)\wedge\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\Rightarrow$
$\{\!|V_1,\ldots,V_k|\!\}_{m^-}\in\rho(z_\bullet)$
the attacker may construct new encryptions using the keys known

(4) $\wedge_{k\in\mathcal{A}_\kappa}\ \forall V_1,\ldots,V_k:\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\Rightarrow\langle V_1,\ldots,V_k\rangle\in\kappa$
the attacker may actively forge new communications

(5) $\{n_\bullet,m_\bullet^{\pm}\}\cup\mathcal{N}_f\subseteq\rho(z_\bullet)$
the attacker initially has some knowledge

Table 4.3: The Attacker's Capabilities

Example 4.6 *Below is given the encoding of the Credit Card Information protocol from Example 4.1, where the key K is not restricted by $(\nu\ K)$.*

$$(\nu\ CCI)\langle A,B,\{CCI\}_K\rangle.0\mid(A,B;y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{;yCCI\}_K\ \mathsf{in}\ 0$$

Since K is not restricted, it becomes one of the free names and hence is known to the attacker, i.c. $K\in\rho(z_\bullet)$, by (5) in Table 4.3. So the attacker is able to decrypt $\{CCI\}_K$, by (2), or generate arbitrary encryption using the key K, by (3), and send it to the principal B, by (4).

4.4 Analysis Results of the Worked Example

A protocol can be deployed in different scenarios, thus when modelling a protocol in LySa, there are various assumptions to make. Depending on which options one runs the analysis with, the analysis will, in most of the cases, return different elements for the analysis components, i.e. ρ and κ. This is due to the choice of 1) allowing the principals to take both the initiator and responder's role or merely one of the roles; and 2) allowing the principals to use different or the

same key to initiate and respond in a session with others. In case the attacker is allowed to behave as a legitimate principal, the different contents of the analysis components may be also due to the possibility that the attacker behaves as a principal and initiated a protocol run with another principal, which may result in the server's decrypting a message encrypted by the attacker at some point.

In section 3.3, the Otway-Rees protocol is used as a worked example to show various encodings according to different assumptions of the protocol's deployment, as listed in Table 3.8 - 3.11. The analysis on those protocol encodings yields the following results. Due to space, only the entries that are not shared by all the four cases are listed below.

Version	Table 3.8	Table 3.9	Table 3.10	Table 3.11
ρ		$\{\lfloor N_{Aij} \rfloor\} \in \rho(znb_{ij})$ $\{\lfloor N_{Bij} \rfloor\} \in \rho(zna_{ij})$		$\{\lfloor N_{Aij} \rfloor\} \in \rho(znb_{ij})$ $\{\lfloor N_{Bij} \rfloor\} \in \rho(zna_{ij})$

Table 4.4: Analysis results of the different encodings of the Otway-Rees protocols

The analysis results do not directly tell anything about the security properties. However, by carefully inspecting the entries, one can conclude that something unintended may happen in the second and the fourth cases: the value $\lfloor N_{Aij} \rfloor$ may be bound to the variable znb_{ij}, which is only intended to be bound to $\lfloor N_{Bij} \rfloor$, and the value $\lfloor N_{Bij} \rfloor$ may be bound to the variable zna_{ij}, which is only intended to be bound to $\lfloor N_{Aij} \rfloor$.

4.5 Why Annotations

Analysing a protocol is not an easy task, it has to do with calculating all the possible behaviours of the protocols, which is, in most cases, uncomputable. So instead of trying to represent the exact behaviour of the protocol, a control flow analysis is employed to only abstract out certain aspects of the protocol behaviours. However, because of the "abstraction", the analysis result cannot be made as precise as the protocol really is. This imprecision means that the analysis only computes over-approximations to the behaviour of the protocol that is being analysed, in which case, absence of a particular element in the analysis components means that the corresponding protocol execution is not part of the behaviour of the protocol. In the context of security, this means that the analysis can be used to verify security properties. However it is not easy to tell directly from the analysis result whether a security property holds or

not, e.g. the analysis results listed in Table 4.4 suggest something unintended may happen in the case of encodings in Table 3.9 and 3.11, but it is almost impossible to draw a conclusion about whether it represents a real attack and which security property has been compromised. Nevertheless, in fact, there do exist type flaw attacks in those cases.

In the following chapters, we shall introduce various kinds of annotations, which syntactically express the intended behaviour of the protocols with respect to different security properties. Consequently, the control flow analysis is extended correspondingly to take the annotations into account, and, as a result, the analysis result will give an explicit sign in case the security property in question has been compromised.

Chapter 5

Authentication

Authentication is an essential security property for security protocols, where communications happen over an insecure network. In such a setting, messages are not necessarily received by the intended principals because an attacker might redirect them or replace them with forged messages. Hence it is very difficult to ensure authentication: is the message received really sent out by the expected principal?

To ensure that communications do take place in the expected way, it is necessary to validate authentication properties of protocols. The authentication study presented in this chapter comes from [20], where the property is referred to as *destination and origin authentication*. The idea is based on the fact that when designing a security protocol, one must have an explicit intension of where messages are supposed to end up. Validating authentication properties then amounts to checking whether messages which came out from some *origin* always have the intended *destination* disregarding the interference of an attacker.

Example 5.1 *Consider the credit card information protocol in Example 3.1, which can be modelled in LySA as the following,*

$$(\nu\ K)(\quad (\nu\ CCI)\langle A, B, \{CCI\}_K\rangle.0 \\ \quad\ |\ \\ \quad (A, B; y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI\}_K\ \mathsf{in}\ 0)$$

One intention of the protocols is to send the encrypted value $\{CCI\}_K$ from A to B, which then decrypts the value and reads the credit card information CCI. Furthermore, B should be able to conclude that the value CCI was indeed sent by principal A.

In the rest of the chapter, we shall briefly recall the control flow analysis for authentication from [15, 16], which makes use of annotations and reference monitors.

5.1 Setting the Scene

Authentication, loosely speaking, can be expressed as "messages should end up at the expected places". Because of the existence of the attacker, who is always able to re-direct any message flowing over the network, this property can not hold in the malicious environment. Rather, it is reasonable to consider only the secret data, which is, in most cases, encrypted before being sent out. Therefore authentication can further be refined as "encryptions should only be decrypted at the expected places" and similarly for decryptions.

To reinforce this condition, the syntax of the LySA calculus is extended to include some annotations, namely *crypto-points*, to express the intended protocol behaviours. More specifically, each encryption is decorated with a *crypto-point*, $c \in CP$ and a set of *crypto-points*, $C \in \mathcal{P}(CP)$. The idea is that, for each encryption the annotations make explicit its *origin*, i.e. the point in the narration where the message is encrypted, and its *destinations*, i.e. the set of the intended points of decryption. Similarly, for the decrypted data, the annotations make explicit the *destination*, i.e. the point in the narration where they are decrypted, c, and their intended *origins*, C, i.e. the set of expected places of encryption. Each crypto-point is a label, which will be associated to a single point of encryption or decryption in the dynamic view of the protocol.

Syntactically, encryption expressions with the annotations are:

$$\{E_1, \ldots, E_k\}_{E_0}[\mathsf{at}\ c\ \mathsf{dest}\ C] \quad \text{and}$$
$$\{\!|E_1, \ldots, E_k|\!\}_{E_0}[\mathsf{at}\ c\ \mathsf{dest}\ C]$$

and, correspondingly, decryption constructs with the intended points of *origin* become

$$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}_{E_0}[\mathsf{at}\ c\ \mathsf{orig}\ C]\ \mathsf{in}\ P \quad \text{and}$$
$$\mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j; x_{j+1}, \ldots, x_k|\!\}_{E_0}[\mathsf{at}\ c\ \mathsf{orig}\ C]\ \mathsf{in}\ P$$

Note that to check the authentication property of a process, its intended behaviour has to be explicitly expressed in terms of the destination and origin annotations, and furthermore, in order for a LySA process to ensure the destination and origin authentication all the requirements placed by the annotations must be met every time a value is decrypted.

Example 5.2 *In the credit card information protocol, the value* $\{CCI\}_K$ *is expected to be decrypted in the decryption point in* B*. This intention can be expressed in the destination and origin annotations as,*

$$(\nu\ CCI)\langle A, B, \{CCI\}_K[\mathsf{at}\ a\ \mathsf{dest}\ \{b\}]\rangle.0$$
$$|$$
$$(A, B; y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI\}_K[\mathsf{at}\ b\ \mathsf{orig}\ \{a\}]\ \mathsf{in}\ 0$$

from A's side: $\{CCI\}_K[\mathsf{at}\ a\ \mathsf{dest}\ \{b\}]$*. The annotation expresses that the* $\{CCI\}_K$ *is encrypted at the place* a *and should ONLY be decrypted at the place* b*.*

from B's side: $\mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI\}_K[\mathsf{at}\ b\ \mathsf{orig}\ \{a\}]\ \mathsf{in}$*. The annotations expresses that this decryption happens at the place* b *and the decrypted value should be encrypted at* a *ONLY.*

Technically, because of the extension of the syntax of the calculus, it is necessary to redefine its semantics as well.

5.2 Dynamic Property

After extending the syntax with *origin* and *destination* annotations, the semantics has to be redefined as well in order to take annotations into account. To this aim, the annotations are carried on to the semantic domain of values. This annotated value domain is referred to as $DVal$, ranged over by DV built from the grammar:

$$\begin{array}{rcl} DV & ::= & n \\ & | & m^+ \\ & | & m^- \\ & | & \{DV_1, \ldots, DV_k\}_{DV_0}[\mathsf{at}\ c\ \mathsf{dest}\ C] \\ & | & \{\!|DV_1, \ldots, DV_k|\!\}_{DV_0}[\mathsf{at}\ c\ \mathsf{dest}\ C] \end{array}$$

Furthermore, an equivalence relation over $DVal$ is defined as $DV_1 = DV_2$ where the annotations associated with encrypted values are ignored. For example, $\{n\}_K[\mathsf{at}\ c_1\ \mathsf{dest}\ C_1] = \{n\}_K[\mathsf{at}\ c_2\ \mathsf{dest}\ C_2]$ for any c_1, c_2, C_1 and C_2.

The semantics of the annotated syntax now is then defined by Table 3.5 in Chapter 3 interpreted over annotated values $DV \in DVal$, except for the symmetric decryption and asymmetric decryptions, as listed in Table 5.1.

$$\text{(Dec)}\quad \frac{\wedge_{i=0}^{k} DV_i = DV_i' \ \wedge\ \mathsf{RM}(c, C, c', C')}{\begin{array}{ll}\mathsf{decrypt} & \{DV_1, \ldots, DV_k\}_{DV_0}[\mathsf{at}\ c\ \mathsf{dest}\ C]\ \mathsf{as}\\ & \{DV_1', \ldots, DV_j'; x_{j+1}, \ldots, x_k\}_{DV_0}[\mathsf{at}\ c'\ \mathsf{orig}\ C']\ \mathsf{in}\ P \rightarrow_{\mathcal{R}}\\ & \qquad P[DV_{j+1}'/x_{j+1}, \ldots, DV_k'/x_k]\end{array}}$$

$$\text{(ADec)}\quad \frac{\wedge_{i=1}^{k} DV_i = DV_i' \ \wedge\ \mathsf{RM}(c, C, c', C')}{\begin{array}{ll}\mathsf{decrypt} & \{\!|DV_1, \ldots, DV_k|\!\}_{m^+}[\mathsf{at}\ c\ \mathsf{dest}\ C]\ \mathsf{as}\\ & \{\!|DV_1', \ldots, DV_j'; x_{j+1}, \ldots, x_k|\!\}_{m^-}[\mathsf{at}\ c'\ \mathsf{orig}\ C']\ \mathsf{in}\ P \rightarrow_{\mathcal{R}}\\ & \qquad P[DV_{j+1}'/x_{j+1}, \ldots, DV_k'/x_k]\end{array}}$$

$$\text{(ASig)}\quad \frac{\wedge_{i=1}^{k} DV_i = DV_i' \ \wedge\ \mathsf{RM}(c, C, c', C')}{\begin{array}{ll}\mathsf{decrypt} & \{\!|DV_1, \ldots, DV_k|\!\}_{m^-}[\mathsf{at}\ c\ \mathsf{dest}\ C]\ \mathsf{as}\\ & \{\!|DV_1', \ldots, DV_j'; x_{j+1}, \ldots, x_k|\!\}_{m^+}[\mathsf{at}\ c'\ \mathsf{orig}\ C']\ \mathsf{in}\ P \rightarrow_{\mathcal{R}}\\ & \qquad P[DV_{j+1}'/x_{j+1}, \ldots, DV_k'/x_k]\end{array}}$$

Table 5.1: Operational Semantics for Authentication Property; $P \rightarrow_{\mathcal{R}} P'$, parametersed on $\mathcal{R}$.

Now the reference monitor semantics $P \rightarrow_{\mathsf{RM}} P'$ takes destination and origin annotations into account and defines RM as,

$$\mathsf{RM}(c, C, c', C') = (c \in C' \ \wedge\ c' \in C)$$

This ensures that the crypto-point of the encrypted value is acceptable at the decryption (i.e. $c \in C'$), and that the crypto-point of the decryption is acceptable for the encryption (i.e. $c' \in C$).

Using this semantics, the property of destination and origin authentication can be defined as follows:

Definition 5.1 (Authentication) A process P ensures *destination and origin authentication* if there are no executions

$$P \rightarrow^* P' \rightarrow_{\mathcal{R}} P''$$

such that $c \notin C'$ or $c' \notin C$ when $P' \rightarrow_{\mathcal{R}} P''$ is derived using (Dec) on

$$\begin{array}{ll} \mathsf{decrypt} & \{DV_1, \ldots, DV_k\}_{DV_0}[\mathsf{at}\ c\ \mathsf{dest}\ C]\ \mathsf{as} \\ & \{DV'_1, \ldots, DV'_j; x_{j+1}, \ldots, x_k\}_{DV'_0}[\mathsf{at}\ c'\ \mathsf{orig}\ C']\ \mathsf{in}\ P \end{array}$$

or using (ADec) or (ASig) on

$$\begin{array}{ll} \mathsf{decrypt} & \{\!|DV_1, \ldots, DV_k|\!\}_{m^+}[\mathsf{at}\ c\ \mathsf{dest}\ C]\ \mathsf{as} \\ & \{\!|DV'_1, \ldots, DV'_j; x_{j+1}, \ldots, x_k|\!\}_{m^-}[\mathsf{at}\ c'\ \mathsf{orig}\ C']\ \mathsf{in}\ P \end{array}$$

or

$$\begin{array}{ll} \mathsf{decrypt} & \{\!|DV_1, \ldots, DV_k|\!\}_{m^-}[\mathsf{at}\ c\ \mathsf{dest}\ C]\ \mathsf{as} \\ & \{\!|DV'_1, \ldots, DV'_j; x_{j+1}, \ldots, x_k|\!\}_{m^+}[\mathsf{at}\ c'\ \mathsf{orig}\ C']\ \mathsf{in}\ P \end{array}$$

It says that a process P ensures authentication property if there is no violation of the annotations in any of its executions.

Next, we shall show how to extend the original analysis from Chapter 4 such that it can be used to check whether a process ensures destination and origin authentication according to Definition 5.1.

5.3 Static Property

In the semantics, the value domain is $DVal$ carrying on the annotations. Similarly, in the authentication analysis, the domain of the analysis components, ρ, κ and ϑ, range over annotated canonical values from $\lfloor DVal \rfloor$ rather than over values from $\lfloor Val \rfloor$. The idea of making use of the origin and destination annotations is to identify the intension of protocol, namely, an encrypted message is ONLY decrypted at the expected places. To capture this, an auxiliary error component, ψ, is added to the authentication analysis, which contains error messages for the possible violations of the authentication property. The error messages have the form of a pair of crypto-points, formally,

$$\psi \subseteq \mathcal{P}(CP \times CP)$$

A pair $(\lfloor c \rfloor, \lfloor c' \rfloor) \in \psi$ means that the message encrypted at the place c is decrypted at other place c' than the intended one, and thus the authentication property may be violated. Note that similarly to names and values, canonical values of crypto-points are required by the authentication analysis.

The authentication analysis extends the original one in Table 4.1 in a way that the annotations are taken into account. This is done by

- The value domain is $DVal$, which is basically Val carrying annotations
- An additional error component ψ is included as another analysis component. The judgement of process becomes

  $$\rho, \kappa, \psi \models P$$

  saying that ρ, κ and ψ are an acceptable analysis estimate of the process P
- The semantics of pattern matching in the extended syntax ignores the annotations associated with values. Consequently, in the authentication analysis annotations are ignored, too.

To allow these to work, the analysis rules were changed a little bit comparing to the original one in order to take the annotations into account. The changed rules are listed in Table 5.3 while others remain the same as in Table 4.1.

The rules for analysing symmetric and asymmetric encryptions (DEnc) and (DAEnc) ensure that the annotations are attached to the values that the encryptions are evaluated to.

The rule (DInp) is identical to the original rule (AInp) except that the analysis of pattern matching ignores the annotations by using a special set inclusion operator, E, defined formally as,

$$\lfloor V \rfloor \,\mathsf{E}\, S \quad \text{if and only if there exists } V' \text{ such that } V = V' \text{ and } V' \in S$$

Note that the equivalent relationship $V_1 = V_2$ is defined over $DVal$ that ignores the annotations.

The decryption rules (DDec) and (DADec) also use the operator E when there is need to ignore the annotations. Additionally, they check whether it is the right place to decrypt the message after a successful decryption. If a violation of the destination and origin annotations may occur, the corresponding encryption and decryption points are recorded in the error component ψ.

5.4 The Attacker

In the setup of $P_{sys} \mid P_\bullet$, the attacker process $P_\bullet$ has to be annotated with respect to the extended syntax. A unique crypto-point $c_\bullet$ is used to indicate the encryption place of the attacker and is used in all the at part of annotations in

$$
\begin{array}{ll}
\text{(DEnc)} & \rho \models \{E_1, \ldots, E_k\}_{E_0}[\mathsf{at}\ c\ \mathsf{dest}\ C] : \vartheta \\
& \quad \text{iff } \wedge_{i=0}^{k}\ \rho \models E_i : \vartheta_i\ \wedge \\
& \qquad \forall U_0, \ldots, U_k : \wedge_{i=0}^{k}\ U_i \in \vartheta_i \Rightarrow \\
& \qquad\quad \{U_1, \ldots, U_k\}_{U_0}[\mathsf{at}\ \lfloor c \rfloor\ \mathsf{dest}\ \lfloor C \rfloor] \in \vartheta \\
\text{(DAEnc)} & \rho \models \{\!|E_1, \ldots, E_k|\!\}_{E_0}[\mathsf{at}\ c\ \mathsf{dest}\ C] : \vartheta \\
& \quad \text{iff } \wedge_{i=0}^{k}\ \rho \models E_i : \vartheta_i\ \wedge \\
& \qquad \forall U_0, \ldots, U_k : \wedge_{i=0}^{k}\ U_i \in \vartheta_i \Rightarrow \\
& \qquad\quad \{\!|U_1, \ldots, U_k|\!\}_{U_0}[\mathsf{at}\ \lfloor c \rfloor\ \mathsf{dest}\ \lfloor C \rfloor] \in \vartheta
\end{array}
$$

$$
\begin{array}{ll}
\text{(DInp)} & \rho, \kappa, \psi \models (E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P \\
& \quad \text{iff } \wedge_{i=1}^{j}\ \rho \models E_i : \vartheta_1\ \wedge \\
& \qquad \forall \langle U_1, \ldots, U_k \rangle \in \kappa : \wedge_{i=1}^{j}\ U_i \,\mathsf{E}\, \vartheta_i \Rightarrow \\
& \qquad\quad (\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor)\ \wedge \rho, \kappa, \psi \models P) \\
\text{(DDec)} & \rho, \kappa, \psi \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}_{E_0}\ \mathsf{in}\ P \\
& \quad \text{iff } \rho \models E : \vartheta\ \wedge\ \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i \wedge \\
& \qquad \forall \{U_1, \ldots, U_k\}_{U_0}[\mathsf{at}\ \lfloor c \rfloor\ \mathsf{dest}\ \lfloor C \rfloor] \in \vartheta\ \wedge \\
& \qquad\quad \wedge_{i=0}^{j}\ U_i \,\mathsf{E}\, \vartheta_i \Rightarrow \\
& \qquad\qquad (\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor)\ \wedge\ \rho, \kappa, \psi \models P\ \wedge \\
& \qquad\qquad\quad (\lfloor c \rfloor \notin \lfloor C' \rfloor\ \vee\ \lfloor c' \rfloor \notin \lfloor C \rfloor \Rightarrow (\lfloor c' \rfloor, \lfloor c \rfloor) \in \psi)) \\
\text{(DADec)} & \rho, \kappa, \psi \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j; x_{j+1}, \ldots, x_k|\!\}_{E_0}\ \mathsf{in}\ P \\
& \quad \text{iff } \rho \models E : \vartheta\ \wedge\ \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i \wedge \\
& \qquad \forall \{\!|U_1, \ldots, U_k|\!\}_{U_0}[\mathsf{at}\ \lfloor c \rfloor\ \mathsf{dest}\ \lfloor C \rfloor] \in \vartheta : \forall U_0' \in \vartheta_0 : \\
& \qquad\quad \forall (\lfloor m^+ \rfloor, \lfloor m^- \rfloor) : \{U_0, U_0'\} = \{m^+, m^-\} \wedge \\
& \qquad\qquad \wedge_{i=1}^{j}\ U_i \,\mathsf{E}\, \vartheta_i \Rightarrow \\
& \qquad\qquad\quad (\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor)\ \wedge\ \rho, \kappa, \psi \models P\ \wedge \\
& \qquad\qquad\qquad (\lfloor c \rfloor \notin \lfloor C' \rfloor\ \vee\ \lfloor c' \rfloor \notin \lfloor C \rfloor \Rightarrow (\lfloor c' \rfloor, \lfloor c \rfloor) \in \psi))
\end{array}
$$

Table 5.2: Authentication Analysis of LySA Terms, $\rho \models E : \vartheta$, and Processes $\rho, \kappa, \psi \models P$. The remaining cases of the analysis overlap the ones in the original analysis as in Table 4.1.

the attacker process. In the orig and dest part of the annotations in the attacker process, a set of all crypto-points CP is used, which means that, in the setup of $P_{sys} \mid P_\bullet$, only the annotations in P_{sys} will cause the authentication property to be violated.

(2) $\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{DV_1,\ldots,DV_k\}_{DV_0}[\mathsf{at}\ c\ \mathsf{dest}\ C] \in \rho(z_\bullet) :$
$DV_0\ \mathsf{E}\ \rho(z_\bullet) \Rightarrow (\wedge_{i=1}^{k} DV_i \in \rho(z_\bullet)\ \wedge\ (\neg RM(c, CP, c_\bullet, C) \Rightarrow (c, c_\bullet) \in \psi))$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{\!|DV_1,\ldots,DV_k|\!\}_{m^+}[\mathsf{at}\ c\ \mathsf{dest}\ C] \in \rho(z_\bullet) :$
$m^-\ \mathsf{E}\ \rho(z_\bullet) \Rightarrow (\wedge_{i=1}^{k} DV_i \in \rho(z_\bullet)\ \wedge\ (\neg RM(c, CP, c_\bullet, C) \Rightarrow (c, c_\bullet) \in \psi))$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{\!|DV_1,\ldots,DV_k|\!\}_{m^-}[\mathsf{at}\ c\ \mathsf{dest}\ C] \in \rho(z_\bullet) :$
$m^+\ \mathsf{E}\ \rho(z_\bullet) \Rightarrow (\wedge_{i=1}^{k} DV_i \in \rho(z_\bullet)\ \wedge\ (\neg RM(c, CP, c_\bullet, C) \Rightarrow (c, c_\bullet) \in \psi))$

the attacker may learn by decrypting messages with keys already known

(3) $\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\ DV_0,\ldots,DV_k : \wedge_{i=0}^{k} DV_i \in \rho(z_\bullet) \Rightarrow$
$\{DV_1,\ldots,DV_k\}_{DV_0}[at\ c_\bullet\ dest\ CP] \in \rho(z_\bullet)$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\ DV_1,\ldots,DV_k : m^+ \in \rho(z_\bullet) \wedge \wedge_{i=1}^{k} DV_i \in \rho(z_\bullet) \Rightarrow$
$\{\!|DV_1,\ldots,DV_k|\!\}_{m^+}[at\ c_\bullet\ dest\ CP] \in \rho(z_\bullet)$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\ DV_1,\ldots,DV_k : m^- \in \rho(z_\bullet) \wedge \wedge_{i=1}^{k} DV_i \in \rho(z_\bullet) \Rightarrow$
$\{\!|DV_1,\ldots,DV_k|\!\}_{m^-}[at\ c_\bullet\ dest\ CP] \in \rho(z_\bullet)$

the attacker may construct new encryptions using the keys known

Table 5.3: The Attacker's Capabilities in Authentication Analysis. The remaining entries of the attacker's capabilities overlap the original ones as in Table 4.3.

Some of the Dolev-Yao condition for the authentication analysis is listed in Table 5.3, while the others remain unchanged as in Table 4.3.

Theorem 5.2 ***(Soundness of Dolev-Yao condition)*** *if (ρ, κ, ψ) satisfies $\mathcal{F}_{RM}^{DY}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$ then $\rho, \kappa \models \overline{Q} : \psi$ for all attackers Q of extended type $(\{z_\bullet\}, \mathcal{N}_{\mathsf{f}} \cup \{n_\bullet\}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$, namely, the canonical definition variables and definition type variables are in $\{z_\bullet\}$ and the canonical names are in $\mathcal{N}_{\mathsf{f}} \cup \{n_\bullet\}$.*

Example 5.3 *Consider the credit card information protocol from Example 4.1. Authentication analysis gives an empty error component ψ even when the process is under attack. This is simply because no destination and origin annotations are specified.*

Example 5.4 *Consider the credit card information protocol with destination and origin annotations from Example 5.2. The annotations formalise the intuition that the encryption at A is supposed to be decrypted at the decryption at*

principal B, only.

$$
\begin{array}{l}
(\nu\ K)(\\
\quad (\nu\ CCI)\langle A, B, \{CCI\}_K[\mathsf{at}\ a\ \mathsf{dest}\ \{b\}]\rangle.0 \\
\quad | \\
\quad (A, B; y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI\}_K[\mathsf{at}\ b\ \mathsf{orig}\ \{a\}]\ \mathsf{in}\ 0
\end{array}
$$

The analysis finds that the authentication property may be violated when the process is put in parallel with the attacker's process, i.e. $P \mid P_\bullet$:

$$\{(a, c_\bullet), (c_\bullet, b)\} \subseteq \psi$$

The first pair $(a, c_\bullet)$ reports that something encrypted at the crypto-point a may be decrypted by an attacker. The second pair $(c_\bullet, b)$ reports that something encrypted by the attacker may end up being decrypted at the crypto-point b. This corresponds to the problems shown in Example 2.2.

5.5 Correctness of the Authentication Analysis

The most interesting part about the authentication analysis is whether it correctly captures violations of the authentication property. This is stated in the following theorem and proved in [20]. We also refer to [20] for a complete list of proofs regarding the correctness the control flow analysis and the authentication analysis.

Theorem 5.3 (Static check for reference monitor) *If $\rho, \kappa, \psi \models P$ and $\psi = \emptyset$ then RM cannot abort P.*

Theorem 5.4 (Analysis of Authentication) *If $\rho, \kappa, \psi \models P$ and $\psi = \emptyset$ then P ensures destination and origin authentication.*

5.6 Meta Level Analysis of Authentication

In a meta level process, indices are consistently added to names and variables such that, say, names, variables used in a session between A_i and B_j are indexed ij, in order to model the scenario in which the meta level process may be deployed. The meta level authentication analysis is identical to the original analysis listed in Table 4.2 in Chapter 4 except that the authentication analysis

listed in Table 5.3 is used to analyse object level constructs. Similar to the object level analysis, in Table 5.3, an error component is included in the meta level authentication analysis, i. e. the analysis is of the form

$$\rho, \kappa, \psi \models_\Gamma M$$

Because each meta level process may instantiate to a set of object level processes, a meta level process ensures destination and origin authentication only if all the processes it instantiates to ensure destination and origin authentication.

Example 5.5 *The credit card information protocol from Example 5.2 may be modelled as a meta level process with annotations, as shown below,*

$$\begin{array}{l}
\mathsf{let}\ X \subseteq S_1\ \mathsf{in} \\
\mathsf{let}\ Y \subseteq S_2\ \mathsf{in} \\
(\nu\ K)(\\
\quad |_{i\in X}|_{j\in Y}(\nu\ CCI_{ij})\langle A_i, B_j, \{CCI_{ij}\}_K[\mathsf{at}\ a\ \mathsf{dest}\ \{b\}]\rangle.0 \\
| \\
\quad |_{i\in X}|_{j\in Y}(A_i, B_j; y_{ij}).\mathsf{decrypt}\ y_{ij}\ \mathsf{as}\ \{; yCCI_{ij}\}_K[\mathsf{at}\ b\ \mathsf{orig}\ \{a\}]\ \mathsf{in}\ 0)
\end{array}$$

The annotations are added to the meta level process saying that something encrypted by principal A_i should only be decrypted by principal B_j. Taking $S_1 = \{1\}$ and $S_2 = \{1\}$ the analysis gives the result $\psi = \emptyset$ even when the process is under attack. This guarantees that any instance of the meta level process ensures the authentication property as specified by the destination and origin annotations.

However, in the above example, all the principals A_i share the same crypto-point a and all the principals B_j share the same crypto-point b. The annotations do not specify by which instance of the principals an encryption is decrypted. To make the authentication property more refined with respect to different scenarios, the crypto-points should be indexed as well. More specifically, crypto-points, c, are now of the form $c_{\bar{i}}$. The indexes in $\bar{i}$ will be substituted whenever the meta level process is instantiated.

Example 5.6 *Consider the object level process from Example 5.2. The meta level process may be modelled as the following, where the crypto-points from different sessions are distinguished by the indices. Assume it is deployed in a scenario where each principal A_i is engaged in communicating with the principal B_i.*

$$
\begin{array}{l}
\mathit{let}\ X \subseteq S\ \mathit{in} \\
(\nu\ K)(\\
\quad |_{i \in X} (\nu\ CCI_i)\langle A_i, B_i, \{CCI_i\}_K[\mathsf{at}\ a_i\ \mathsf{dest}\ \{b_i\}]\rangle.0 \\
| \\
\quad |_{i \in X} (A_i, B_i; y_i).\mathsf{decrypt}\ y_i\ \mathsf{as}\ \{; yCCI_i\}_K[\mathsf{at}\ b_i\ \mathsf{orig}\ \{a_i\}]\ \mathit{in}\ 0)
\end{array}
$$

Here annotations have been added stating that an instance of the initiator, A_i, uses the crypto-point a_i for the encryptions it makes for communicating with the instance of the responders, B_i. Taking $\lfloor S \rfloor = \{1, 2\}$, the analysis finds that

$$\{(a_1, b_2), (a_2, b_1)\} \subseteq \psi$$

Thus, the analysis reports possible attacks because something encrypted by A_1 may be wrongfully decrypted by B_2 and something encrypted by A_2 may be wrongfully decrypted by B_1. This is really a violation of the property, because one can find an execution that leads to the above violations of the authentication property. Consider the following message exchange of two sessions:

	A_1	A_2	B_2	*Attacker*
1.1	$\langle A_1, B_1, \{CCI_1\}_K$ $[\mathsf{at}\ a_1\ \mathsf{dest}\ \{b_1\}]\rangle$			$(; z_{A_1}, z_{B_1}, z_{enc1})$
2.1		$\langle A_2, B_2, \{CCI_2\}_K$ $[\mathsf{at}\ a_2\ \mathsf{dest}\ \{b_2\}]\rangle$		$(; z_{A_2}, z_{B_2}, z_{enc2})$
2.1′			$(A_2, B_2; y_2)$	$\langle z_{A_2}, z_{B_2}, z_{enc1}\rangle$

After the last message exchange, variable y_2 has become bound to the value of z_{enc1}, i.e. to the value $\{CCI_1\}_K[\mathsf{at}\ a_1\ \mathsf{dest}\ \{b_1\}]$. The following decryption and pattern matching can then successfully take place

$$\mathsf{decrypt}\ \{CCI_1\}_K[\mathsf{at}\ a_1\ \mathsf{dest}\ \{b_1\}]\ \mathsf{as}\ \{; yCCI_2\}_K[\mathsf{at}\ b_2\ \mathsf{orig}\ \{a_2\}]\ \mathit{in}\ 0 \rightarrow 0$$

This decryption violates the authentication property and represents an attack because something encrypted at principal A_1 is wrongfully decrypted at principal B_2. This accounts for the error (a_1, b_2) in ψ.

5.7 Summary

Authentication exists at many different levels of abstraction. At a lower level, cryptographic primitives are employed by message authentication code. At a higher level, one may validate on authentication property by reasoning about the beliefs of principals. In the work presented in this chapter, the authentication property is described at the level of the individual messages used in communications, and in particular, the part of the messages that is safeguarded by cryptography. Authentication is assured when there is no violation of the *destination and origin* annotations. However, a violation of the property does not necessarily mean the overall aims of the protocol have been compromised. Rather, it points to potential vulnerabilities that should become subject of closer inspection. The usefulness of the *destination and origin* annotations has been demonstrated by applying the authentication analysis to a number of classical protocols, e.g. Andrew Secure RPC protocol, Needham-Schroeder Symmetric Key protocol, as well as real world protocol, e.g. PKMv2 Protocol in IEEE 802.16e-2005 specification. All the results show that the analysis is able to reason about the authentication properties of the protocols by either revealing authentication based vulnerabilities or confirming the absence of such vulnerabilities.

Chapter 6

Confidentiality

An important security property concerning cryptographic protocols is confidentiality. Confidentiality ensures that data is only available to those authorised to obtain it. This is usually achieved through encryption of the data so that only those with the correct decryption key can recover it. In cryptographic protocols confidentiality is essential to ensure that keys and other data are available only as intended, meaning that those important data should only be obtained by the *right* party. This intension can be achieved by clarifying the intended place where the data goes to, which, in our work, is expressed as confidentiality annotations.

Example 6.1 *Consider the credit card information protocol in Example 3.1, which can be modelled in LySA as the following,*

$$\begin{array}{l}((\nu\ CCI)\langle A, B, \{CCI\}_K\rangle.0\\ \mid\\ (A, B; y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI\}_K\ \mathsf{in}\ 0)\end{array}$$

Again the scope of key K is not restricted such that it is known to the attacker. By intercepting the encrypted value and decrypting it, the attacker will learn the value of CCI. In this sense, the confidentiality of this protocol can be expressed as the value CCI should only be bound to the variable $yCCI$.

6.1 Setting the Scene

Confidentiality is about protecting sensitive data from unauthorised parties. Because of the existence of the attacker, who has total control of the network, legitimate protocol participants may be fooled into revealing some secret data to the attacker or to accepting fake data as the real one. In this sense, confidentiality is about "whether secret data ONLY reaches the intended place" and, the other way around, "whether secret data comes from the intended place". Therefore confidentiality expressed in LySA can be further refined as "whether a value is bound to the right variable" and "whether a variable is the right one to bind the value to". It is reasonable to consider only the secret data, which is, in most cases, encrypted before being sent out.

To describe the confidentiality intentions of protocols in LySA, whenever a name is introduced we extend LySA with the annotation [within $\mathcal{X}$], where $\mathcal{X} \subseteq \lfloor Var \rfloor$ is a set of canonical variables to which a name may be bound and at each binding occurrence we add the annotations [from $\mathcal{N}$], where $\mathcal{N} \subseteq \lfloor Val \rfloor$ is a set of canonical values that may be bound to a variable.

Syntactically, whenever we generate a new name, we have a LySA term of the form

$$(\nu\ n[\text{within } \mathcal{X}])P \qquad \text{and}$$
$$(\nu_{\pm}\ m[\text{within } \mathcal{X}_1, \mathcal{X}_2])P$$

where $\mathcal{X}$, $\mathcal{X}_1$ and $\mathcal{X}_2$ state the sets of variables to which the name n, the public key m^+ and the private key m^- may be bound, respectively.

Similarly, variables occurring in binding places inside a decryption are also decorated with annotations. The decryption constructs are of the form,

$$\text{decrypt } E \text{ as } \{E_1, \ldots, E_j; x_{j+1}[\text{from } \mathcal{N}_{j+1}], \ldots, x_k[\text{from } \mathcal{N}_k]\}^l_{E_0} \text{ in } P \quad \text{and}$$
$$\text{decrypt } E \text{ as } \{|E_1, \ldots, E_j; x_{j+1}[\text{from } \mathcal{N}_{j+1}], \ldots, x_k[\text{from } \mathcal{N}_k]|\}^l_{E_0} \text{ in } P$$

where $l \in Lab$ is a label uniquely identifying each decryption place.

Example 6.2 *In the credit card information protocol from Example 6.1, the value CCI is expected to be bound to the variable $yCCI$ only and the variable*

yCCI is expected to be bound to the value CCI. These intensions can be expressed in the within *and* from *annotations as,*

$$\begin{array}{l}((\nu\ CCI[\textsf{within}\ \{yCCI\}])\langle A, B, \{CCI\}_K\rangle.0\\ |\\ (A, B; y).\textsf{decrypt}\ y\ \textsf{as}\ \{; yCCI[\textsf{from}\ \{CCI\}]\}^{l}_{K}\ \textsf{in}\ 0)\end{array}$$

from A's side $(\nu\ CCI[\textsf{within}\ \{yCCI\}])$*. The annotation expresses that the value CCI should only be bound to the variable yCCI, and*

from B's side $\textsf{decrypt}\ y\ \textsf{as}\ \{; yCCI[\textsf{from}\ \{CCI\}]\}_K\ \textsf{in}\ 0$*. The annotation expresses that CCI is the only value that should be bound to the variable yCCI.*

Technically, because of the extension of the syntax of the LySA calculus, it is necessary to redefine the semantics as well.

6.2 Dynamic Property

The semantics of the LySA calculus remains unchanged after the extension. The confidentiality annotations are not carried around, but rather recorded in an abstract environment, β, which is used to collect the within annotations associated with each name,

$$\beta : \mathcal{N} \rightarrow \mathcal{P}(Var)$$

The reduction relation now holds between a pair of processes, written

$$\beta \vdash P \rightarrow P'$$

precisely P can evolve into P' when confidentiality annotations of names are as described in the environment β.

The within and from annotations are enforced by a reference monitor, which aborts unwanted variable bindings, the rules for symmetric and asymmetric restrictions and decryptions are as shown in Table 6.1, the rest of the rules are similar to the ones in the original semantics (Table 3.5) except that the abstract environment β is taken into account. The reference monitor makes use of a function $\mathcal{I}$ to determine the annotation of each value. Recall that values $V \in Val$ are built from the grammar

$$V ::= n \mid m^+ \mid m^- \mid \{V_1, \ldots, V_k\}_{V_0} \mid \{\!|V_1, \ldots, V_k|\!\}_{V_0}$$

Definition 6.1 Determine within Annotations

$$\begin{array}{l} \mathcal{I} : Val \to \mathcal{P}(Var) \\ \quad \begin{array}{lcl} \mathcal{I}(n) & = & \beta(n) \\ \mathcal{I}(m^+) & = & \beta(m^+) \\ \mathcal{I}(m^-) & = & \beta(m^-) \\ \mathcal{I}(\{V_1, \ldots, V_k\}_{V_0}) & = & \mathcal{X} \\ \mathcal{I}(\{\!|V_1, \ldots, V_k|\!\}_{V_0}) & = & \mathcal{X} \end{array} \end{array}$$

Note encryptions are implicitly allowed to become bound to all the variables, $\mathcal{X}$, that occur in the process.

(Dec)
$$\frac{\wedge_{i=0}^{j} V_i = V_i' \ \wedge \ \wedge_{i=j+1}^{k} \mathsf{RM}(V_i, \mathcal{I}(V_i), x_i, \mathcal{N}_i)}{\begin{array}{l} \beta \vdash \mathsf{decrypt} \quad \{V_1, \ldots, V_k\}_{V_0} \ \mathsf{as} \\ \qquad\qquad \{V_1', \ldots, V_j'; x_{j+1}[\mathsf{from}\ \mathcal{N}_{j+1}], \ldots, x_k[\mathsf{from}\ \mathcal{N}_k]\}_{V_0'}^{l} \\ \mathsf{in}\ P \to_{\mathcal{R}} P[V_{j+1}'/x_{j+1}, \ldots, V_k'/x_k] \end{array}}$$

(ADec)
$$\frac{\wedge_{i=1}^{j} V_i = V_i' \ \wedge \ \wedge_{i=j+1}^{k} \mathsf{RM}(V_i, \mathcal{I}(V_i), x_i, \mathcal{N}_i)}{\begin{array}{l} \beta \vdash \mathsf{decrypt} \quad \{\!|V_1, \ldots, V_k|\!\}_{m^+} \ \mathsf{as} \\ \qquad\qquad \{\!|V_1', \ldots, V_j'; x_{j+1}[\mathsf{from}\ \mathcal{N}_{j+1}], \ldots, x_k[\mathsf{from}\ \mathcal{N}_k]|\!\}_{m^-}^{l} \\ \mathsf{in}\ P \to_{\mathcal{R}} P[V_{j+1}'/x_{j+1}, \ldots, V_k'/x_k] \end{array}}$$

(ASig)
$$\frac{\wedge_{i=1}^{j} V_i = V_i' \ \wedge \ \wedge_{i=j+1}^{k} \mathsf{RM}(V_i, \mathcal{I}(V_i), x_i, \mathcal{N}_i)}{\begin{array}{l} \beta \vdash \mathsf{decrypt} \quad \{\!|V_1, \ldots, V_k|\!\}_{m^-} \ \mathsf{as} \\ \qquad\qquad \{\!|V_1', \ldots, V_j'; x_{j+1}[\mathsf{from}\ \mathcal{N}_{j+1}], \ldots, x_k[\mathsf{from}\ \mathcal{N}_k]|\!\}_{m^+}^{l} \\ \mathsf{in}\ P \to_{\mathcal{R}} P[V_{j+1}'/x_{j+1}, \ldots, V_k'/x_k] \end{array}}$$

(New)
$$\frac{\beta[n \mapsto \mathcal{X}] \vdash P \to_{\mathcal{R}} P'}{\beta \vdash (\nu\ n[\mathsf{within}\ \mathcal{X}])P \to_{\mathcal{R}} (\nu\ n[\mathsf{within}\ \mathcal{X}])P'}$$

(ANew)
$$\frac{\beta[m^+ \mapsto \mathcal{X}_1, m^- \mapsto \mathcal{X}_2] \vdash P \to_{\mathcal{R}} P'}{\beta \vdash (\nu_{\pm}\ m[\mathsf{within}\ \mathcal{X}_1, \mathcal{X}_2])P \to_{\mathcal{R}} (\nu_{\pm}\ m[\mathsf{within}\ \mathcal{X}_1, \mathcal{X}_2])P'}$$

Table 6.1: Operational Semantics $\beta \vdash P \to_{\mathcal{R}} P'$, parameterized on $\mathcal{R}$

The operational Semantics $\beta \vdash P \to_{\mathcal{R}} P'$ with the relation $\mathcal{R}$ is considered in

two variants;

- the standard semantics, $\beta \vdash P \rightarrow P'$, discards the annotations and takes $\mathcal{R}(n, \mathcal{X}, x, \mathcal{N})$ to be universally true.
- the reference monitor semantics, $\beta \vdash P \rightarrow_{RM} P'$, takes advantage of the annotations and takes

  $$\mathsf{RM}(n, \mathcal{X}, x, \mathcal{N}) = n \in \mathcal{N} \ \wedge \ x \in \mathcal{X}$$

  Therefore at each binding occurrence, the reference monitor checks whether it is allowed and captures any binding violations with respect to the within and from annotations.

The rules (Dec), (ADec) and (ASig) in Table 6.1 require that in a successful decryption, either symmetric or asymmetric, the values $V_{j+1}, \ldots, V_k$ have to be allowed to be bound the corresponding variables $x_{j+1}, \ldots, x_k$ by the within annotations of the values, and the variables have to be allowed to be binding to the corresponding values by the from annotations. Therefore in the reference monitor semantics no within and from annotation is violated.

The rules (New) and (ANew) update β to include the within annotations of the names and asymmetric key pairs before proceeding.

Using this semantics, the property of within and from confidentiality can be defined as follows:

Definition 6.2 (Confidentiality) A process P ensures *within and from confidentiality* if for all the executions

$$\beta \vdash P \rightarrow^* P' \rightarrow P''$$

there exists

$$\beta \vdash P \rightarrow^* P' \rightarrow_{\mathsf{RM}} P''$$

This definition states that a process P, given the abstract environment β, ensures the confidentiality property if none of its executions is aborted when the reference monitor is turned on. More precisely, there does not exist i $(j+1 \leq i \leq k) : V_i \notin \mathcal{N}_i$ or $x_i \notin \mathcal{I}(V_i)$ when $P' \rightarrow P''$ is derived using (Dec) on

$$\begin{array}{ll} \mathsf{decrypt} & \{V_1, \ldots, V_k\}_{V_0} \ \mathsf{as} \\ & \{V_1', \ldots, V_j'; x_{j+1}[\mathsf{from}\ \mathcal{N}_{j+1}], \ldots, x_k[\mathsf{from}\ \mathcal{N}_k]\}^l_{V_0'} \ \mathsf{in}\ P \end{array}$$

or using (ADec) or (ASig) on

$$\begin{array}{ll}\mathsf{decrypt} & \{\!|V_1,\ldots,V_k|\!\}_{m^+} \text{ as} \\ & \{\!|V_1',\ldots,V_j';x_{j+1}[\text{from } \mathcal{N}_{j+1}],\ldots,x_k[\text{from } \mathcal{N}_k]|\!\}^l_{m^-} \text{ in } P\end{array}$$

or

$$\begin{array}{ll}\mathsf{decrypt} & \{\!|V_1,\ldots,V_k|\!\}_{m^-} \text{ as} \\ & \{\!|V_1',\ldots,V_j';x_{j+1}[\text{from } \mathcal{N}_{j+1}],\ldots,x_k[\text{from } \mathcal{N}_k]|\!\}^l_{m^+} \text{ in } P\end{array}$$

Next, we shall show how to extend the original analysis from Chapter 4 such that it can be used to check whether a process ensures within and from confidentiality according to Definition 6.2.

6.3 Static Property

To capture violations of confidentiality annotations, similar to the authentication analysis, we make use of an error component ψ to collect all the error messages, which is a label indicating where the violation happens. Formally,

$$\psi \subseteq \mathcal{P}(\lfloor Lab \rfloor)$$

A label $l \in \psi$ means that a value binding inside the decryption, marked with label l, violates the confidentiality annotations and therefore is not allowed.

The confidentiality analysis extends the original analysis in a way that the within and from annotations are taken into account. The extended rules are listed in Table 6.2, while the rest of the rules are similar to the original ones in Table 4.1 except that the abstract environment β is added as an additional analysis component.

The rules for symmetric decryption (CDec) and asymmetric decryption (CADec) use pattern matching and variable binding as before, but they further check whether the bindings are allowed, or record the label l in the ψ component.

The rule for restrictions (CNew) and (CANew) require that the declared within annotations for n, m^+ and m^- are included in the abstract environment β.

(CDec) $\rho, \kappa, \beta, \psi \models$
$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}[\mathsf{from}\ \mathcal{N}_{j+1}], \ldots, x_k[\mathsf{from}\ \mathcal{N}_k]\}^l_{E_0}\ \mathsf{in}\ P$
$\mathsf{iff}\ \rho \models E : \vartheta\ \wedge\ \wedge^j_{i=0}\ \rho \models E_i : \vartheta_i \wedge$
$\forall \{U_1, \ldots, U_k\}_{U_0} \in \vartheta\ \wedge$
$\wedge^j_{i=0}\ U_i \in \vartheta_i \Rightarrow$
$(\wedge^k_{i=j+1}\ U_i \in \rho(\lfloor x_i \rfloor)\ \wedge\ \rho, \kappa, \beta, \psi \models P\ \wedge$
$(\wedge^k_{i=j+1}(U_i \notin \lfloor \mathcal{N}_i \rfloor\ \vee\ \lfloor x_i \rfloor \notin \beta(U_i)) \Rightarrow \lfloor l \rfloor \in \psi))$

(CADec) $\rho, \kappa, \beta, \psi \models$
$\mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j; x_{j+1}[\mathsf{from}\ \mathcal{N}_{j+1}], \ldots, x_k[\mathsf{from}\ \mathcal{N}_k]|\!\}^l_{E_0}\ \mathsf{in}\ P$
$\mathsf{iff}\ \rho \models E : \vartheta\ \wedge\ \wedge^j_{i=0}\ \rho \models E_i : \vartheta_i \wedge$
$\forall \{\!|U_1, \ldots, U_k|\!\}_{U_0} \in \vartheta : \forall U_0' \in \vartheta_0 :$
$\forall (m^+, m^-) : \{U_0, U_0'\} = \{\lfloor m^+ \rfloor, \lfloor m^- \rfloor\} \wedge$
$\wedge^j_{i=1}\ U_i \in \vartheta_i \Rightarrow$
$(\wedge^k_{i=j+1}\ U_i \in \rho(\lfloor x_i \rfloor)\ \wedge\ \rho, \kappa, \beta, \psi \models P\ \wedge$
$(\wedge^k_{i=j+1}(U_i \notin \lfloor \mathcal{N}_i \rfloor\ \vee\ \lfloor x_i \rfloor \notin \beta(U_i)) \Rightarrow \lfloor l \rfloor \in \psi))$

(CNew) $\rho, \kappa, \beta, \psi \models (\nu\ n[\mathsf{within}\ \mathcal{X}])P$
$\mathsf{iff}\ \lfloor \mathcal{X} \rfloor \subseteq \beta(\lfloor n \rfloor)\ \wedge\ \rho, \kappa, \beta, \psi \models P$

(CANew) $\rho, \kappa, \beta, \psi \models (\nu_\pm\ m[\mathsf{within}\ \mathcal{X}])P$
$\mathsf{iff}\ \lfloor \mathcal{X} \rfloor \subseteq \beta(\lfloor m^+ \rfloor)\ \wedge\ \lfloor \mathcal{X} \rfloor \subseteq \beta(\lfloor m^- \rfloor)\ \wedge\ \rho, \kappa, \beta, \psi \models P$

Table 6.2: Confidentiality Analysis of LySA Terms, $\rho \models E : \vartheta$, and Processes, $\rho, \kappa, \beta, \psi \models P$. The remaining rules overlap with the original ones as in Table 4.1 but using the judgement form of the confidentiality analysis.

6.4 Correctness of the Confidentiality Analysis

The confidentiality analysis of LySA defined in Table 6.2 statically predicts some aspects of the behaviour of a process. The main result that will be shown in the section is that the analysis of the LySA calculus with within and from annotations does indeed capture the entire behaviour of the process, and more importantly, the confidentiality analysis correctly captures violations to the confidentiality property.

Theorem 6.3 (Correctness of Confidentiality Analysis) *if* $\rho, \kappa, \beta, \psi \models P$ *and* $\psi = \emptyset$ *then* P *ensures* within *and* from *confidentiality.*

PROOF. The theorem can be proven by showing an extended subject reduction result that says if $\rho, \kappa, \beta, \psi \models P$ and $P \to P'$ then $\rho, \kappa, \beta, \psi \models P'$, and furthermore if $\psi = \emptyset$ then $P \to P'$ does not violate the confidentiality property. An induction in the length of the execution sequences then gives that P ensures confidentiality for all executions.

In case (Com) we assume

$$\rho, \kappa, \beta, \psi \models \langle V_1, \ldots, V_k \rangle.P \mid (V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k).Q$$

which amounts to:

(a) $\wedge_{i=1}^{k} \rho \models V_i : \vartheta_i$
(b) $\forall U_1, \ldots, U_k : \wedge_{i=1}^{k} U_i \in \vartheta_i \Rightarrow \langle U_1, \ldots, U_k \rangle \in \kappa$
(c) $\rho, \kappa, \beta, \psi \models P$
(d) $\wedge_{i=1}^{j} \rho \models V_i' : \vartheta_i'$
(e) $\forall \langle U_1, \ldots, U_k \rangle \in \kappa : \wedge_{i=1}^{j} U_i \in \vartheta_i' \Rightarrow (\wedge_{i=j+1}^{k} U_i \in \rho(\lfloor x_i \rfloor) \ \wedge \ \rho, \kappa, \beta, \psi \models Q)$

Moreover we assume that $\wedge_{i=1}^{j} V_i = V_i'$ because
$\langle V_1, \ldots, V_k \rangle.P \mid (V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k).Q \to P \mid Q[V_{j+1}/x_{j+1}, \ldots, V_k/x_k]$ and we have to prove $\rho, \kappa, \beta, \psi \models P \mid Q[V_{j+1}/x_{j+1}, \ldots, V_k/x_k]$. From (a) we have $\wedge_{i=1}^{k} V_i \in \vartheta_i$ since $\wedge_{i=1}^{k} \mathsf{fv}(V_i) = \emptyset$ and then (b) gives $\langle V_1, \ldots, V_k \rangle \in \kappa$. From (d) and the assumption $\wedge_{i=1}^{j} V_i = V_i'$ we get $\wedge_{i=1}^{j} V_i \in \vartheta_i'$. Now (e) gives $\wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor)$ and $\rho, \kappa, \beta, \psi \models Q$. The substitution result **??** then gives $\rho, \kappa, \beta, \psi \models Q[V_{j+1}/x_{j+1}, \ldots, V_k/x_k]$ and together with (c) this gives the required result.

The second part is trivial: when $\psi = \emptyset$, obviously
$\langle V_1, \ldots, V_k \rangle.P \mid (V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k).Q \to_{\mathsf{RM}} P \mid Q[V_{j+1}/x_{j+1}, \ldots, V_k/x_k]$

In case (Dec) we assume

$$\rho, \kappa, \beta, \psi \models \mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}\ \mathsf{as}\ \{V_1', \ldots, V_j'; x_{j+1}[\mathsf{from}\ \mathcal{N}_{j+1}], \ldots, x_k[\mathsf{from}\ \mathcal{N}_k]\}_{V_0'}^{l}\ \mathsf{in}\ P$$

which amounts to:

(f) $\wedge_{i=0}^{k}\ \rho \models V_i : \vartheta_i$
(g) $\forall U_0, \ldots, U_k : \wedge_{i=0}^{k} U_i \in \vartheta_i \Rightarrow \{U_1, \ldots, U_k\}_{U_0} \in \vartheta$
(h) $\wedge_{i=0}^{j} \rho \models V_i' : \vartheta_i'$
(i) $\forall \{U_1, \ldots, U_k\}_{U_0} \in \vartheta : \wedge_{i=0}^{j} U_i \in \vartheta_i'$
$\Rightarrow (\wedge_{i=j+1}^{k} U_i \in \rho(\lfloor x_i \rfloor) \wedge$
$(\wedge_{i=j+1}^{k} (U_i \notin \lfloor \mathcal{N}_i \rfloor \ \vee \ \lfloor x_i \rfloor \notin \beta(U_i)) \Rightarrow \lfloor l \rfloor \in \psi) \ \wedge \rho, \kappa, \beta, \psi \models P)$

Furthermore we assume that $\wedge_{i=0}^{j} V_i = V_i'$ because
$\mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}\ \mathsf{as}\ \{V_1', \ldots, V_j'; x_{j+1}[\mathsf{from}\ \mathcal{N}_{j+1}], \ldots, x_k[\mathsf{from}\ \mathcal{N}_k]\}_{V_0'}^{l}\ \mathsf{in}\ P \to$

$P[V_{j+1}/x_{j+1}, \ldots, V_k/x_k]$ and we have to prove $\rho, \kappa, \beta, \psi \models P[V_{j+1}/x_{j+1}, \ldots, V_k/x_k]$. From (f) and $\wedge_{i=0}^{k} \mathsf{fv}(V_i) = \emptyset$, we get $\wedge_{i=0}^{k} V_i \in \vartheta_i$ and then (g) gives $\{V_1, \ldots, V_k\}_{V_0} \in \vartheta$. From (h) and the assumption $\wedge_{i=0}^{j} V_i = V_i'$ we get $\wedge_{i=0}^{j} V_i \in \vartheta_i'$. Now (i) gives $\wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor)$ and $\rho, \kappa, \beta, \psi \models P$. Using Lemma **??** we get the required result $\rho, \kappa, \beta, \psi \models P[V_{j+1}/x_{j+1}, \ldots, V_k/x_k]$

For the second part of the result we observe that $\wedge_{i=j+1}^{k}(U_i \notin \lfloor \mathcal{N}_i \rfloor \ \vee \ \lfloor x_i \rfloor \notin \beta(U_i)) \Rightarrow \lfloor l \rfloor \in \psi$ follows from (i) and since $\psi = \emptyset$ it must be the case that $\wedge_{i=j+1}^{k} RM(V_i, \mathcal{I}(V_i), x_i, \mathcal{N}_i)$. Thus the condition of the rule (Dec) are fulfilled for $\rightarrow_{\mathsf{RM}}$.

In case (ADec) and (ASig) are similar.

In case (New) we assume $\rho, \kappa, \beta, \psi \models (\nu\ n[\mathsf{within}\ \mathcal{X}])P$, which amounts to:

(a) $\lfloor \mathcal{X} \rfloor \subseteq \beta(n)$
(b) $\rho, \kappa, \beta, \psi \models P$

Furthermore we assume that $\beta[n \mapsto \mathcal{X}] \vdash P \rightarrow Q$. By applying the induction hypothesis on (b), we have $\rho, \kappa, \beta, \psi \models Q$, which together with (a) gives the expected result that $\rho, \kappa, \beta, \psi \models (\nu\ n[\mathsf{within}\ \mathcal{X}])Q$.

In case (ANew) is similar.

In cases (Par) and (Rep) follow directly from the induction hypothesis.

The case (Congr) also uses the congruence result.

6.5 The Attacker

In the setup of $P_{sys} \mid P_\bullet$, the attacker process $P_\bullet$ has to be annotated with respect to the extended syntax. A unique label $l_\bullet$ is used to indicate the decryption places that are used by the attacker. In the annotations in the attacker process, a set of all the names in the system process, $\mathcal{N} = \mathcal{N}_\mathsf{f}$ is used. The variables used by the attacker, $z_\bullet$, are allowed to become bound to all the names. Furthermore, we allow the name $n_\bullet$ generated by the attacker to be bound to all the variables $\mathcal{X}$, i.e. $\beta(n_\bullet) = \mathcal{X}$. Therefore, only the annotations in P_{sys} will cause the confidentiality property to be violated.

Some of the Dolev-Yao condition for the confidentiality analysis is listed in Table 6.3, while the others remain unchanged as in Table 4.3.

(2) $\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{V_1,\ldots,V_k\}_{V_0}\in\rho(z_\bullet):$ $V_0\in\rho(z_\bullet)\Rightarrow(\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\ \wedge\vee_{i=1}^{k}\neg RM(V_i,\beta(V_i),z_\bullet,\mathcal{N})\Rightarrow l_\bullet\in\psi)$ $\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{\!|V_1,\ldots,V_k|\!\}_{m^+}\in\rho(z_\bullet):$ $m^-\in\rho(z_\bullet)\Rightarrow(\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\ \wedge\vee_{i=1}^{k}\neg RM(V_i,\beta(V_i),z_\bullet,\mathcal{N})\Rightarrow l_\bullet\in\psi)$ $\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{\!|V_1,\ldots,V_k|\!\}_{m^-}\in\rho(z_\bullet):$ $m^+\in\rho(z_\bullet)\Rightarrow(\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\ \wedge\vee_{i=1}^{k}\neg RM(V_i,\beta(V_i),z_\bullet,\mathcal{N})\Rightarrow l_\bullet\in\psi)$ the attacker may learn by decrypting messages with keys already known

Table 6.3: The Attacker's Capabilities in Confidentiality Analysis

Similar to the way described in Chapter 4.3, a formula $\mathcal{F}_{RM}^{DY}$ of type $(\mathcal{N}_{\mathsf{f}},\mathcal{A}_\kappa,\mathcal{A}_{\mathsf{Enc}})$ is defined as the conjunction of the five components in Table 4.3 and Table 6.2, where each describes an ability of the attacker. Furthermore, it was proved that the formula $\mathcal{F}_{RM}^{DY}$ is capable of characterising the potential effect of all attackers $P_\bullet$ of type $(\mathcal{N}_{\mathsf{f}},\mathcal{A}_\kappa,\mathcal{A}_{\mathsf{Enc}})$. The soundness of the Dolev-Yao condition is established by the following Theorem.

Theorem 6.4 (Correctness of Dolev-Yao Condition) *If (ρ,κ,β,ψ) satisfies $\mathcal{F}_{RM}^{DY}$ of type $(\mathcal{N}_{\mathsf{f}},\mathcal{A}_\kappa,\mathcal{A}_{\mathsf{Enc}})$ then $\rho,\kappa,\beta,\psi\models\overline{Q}$ for all attackers Q of extended type $(\{z_\bullet\},\mathcal{N}_{\mathsf{f}}\cup\{n_\bullet\},\mathcal{A}_\kappa,\mathcal{A}_{\mathsf{Enc}})$.*

Proof. By structural induction on $\overline{Q}$.

In case of (Inp), i.e. $\overline{Q}$ is the process of the form

$$(\overline{E_1},\ldots,\overline{E_j};x_{j+1}[\mathsf{from}\ \mathcal{N}],\ldots,x_k[\mathsf{from}\ \mathcal{N}]).\overline{P}$$

and here we need to find $\vartheta_1,\ldots,\vartheta_j$ and show

(*a*) $\wedge_{i=1}^{j}\rho\models\overline{E_i}:\vartheta_i$ and
for all $\langle V_1,\ldots,V_k\rangle\in\kappa$ with $\wedge_{i=1}^{j}V_i\in\vartheta_i$ that:
(*b*) $\wedge_{i=j+1}^{k}V_i\in\rho(\lfloor x_i\rfloor)$
(*c*) $\rho,\kappa,\beta,\psi\models\overline{P}$

We choose $\vartheta_i(1\leq i\leq j)$ as the least set such that $\rho\models\overline{E_i}:\vartheta_i$ and prove that $\vartheta_i\subseteq\rho(z_\bullet)$; intuitively, if $\overline{E_i}$ has free variables $z_1,\ldots,z_m$ then ϑ_i consists of all values $\lfloor\overline{E_i}[V_1/z_1,\ldots,V_m/z_m]\rfloor$ where $V_j(1\leq j\leq m)\in\rho(z_\bullet)$, and this takes care of (*a*). Next consider $\langle V_1,\ldots,V_k\rangle\in\kappa$ and assume that $\wedge_{i=1}^{j}V_i\in\vartheta_i$.

Since $\wedge_{i=1}^{j}\vartheta_i \subseteq \rho(z_\bullet)$, as above, we have $\wedge_{i=1}^{j}V_i \in \rho(z_\bullet)$ and by $\mathcal{F}_{RM}^{DY}$ we get $\wedge_{i=j+1}^{k}V_i \in \rho(z_\bullet)$ Since $\lfloor x_i \rfloor = z_\bullet$ this takes care of (b); furthermore $\overline{P}$ has extended type $(\{z_\bullet\}, \mathcal{N}_f \cup \{n_\bullet\}, \mathcal{A}_\kappa, \mathcal{A}_{ENC})$ and the induction hypothesis then takes care of (c).

In case of (Dec), i.e. $\overline{Q}$ is the process of the form

$$\mathsf{decrypt}\ \overline{E}\ \mathsf{as}\ \{\overline{E_1}, \ldots, \overline{E_j}; x_{j+1}[\mathsf{from}\ \mathcal{N}], \ldots, x_k[\mathsf{from}\ \mathcal{N}]\}_{\overline{E_0}}^{l_\bullet}\ \mathsf{in}\ \overline{P}$$

and we need to find ϑ and $\vartheta_0, \ldots, \vartheta_j$ and show

(a) $\rho \models \overline{E} : \vartheta \ \wedge\ \wedge_{i=0}^{j}\rho \models \overline{E_i} : \vartheta_i$
and for all $\{V_1, \ldots, V_k\}_{V_0}^{l} \in \vartheta$ with $\wedge_{i=0}^{j}V_i \in \vartheta_i$ that:
(b) $\wedge_{i=j+1}^{k}V_i \in \rho(x_i)$
(c) $\wedge_{i=j+1}^{k}\neg\mathsf{RM}(V_i, \mathcal{I}(V_i), x_i, \mathcal{N}) \Rightarrow l_\bullet \in \psi$
(d) $\rho, \kappa, \beta, \psi \models \overline{P} : \psi$

We choose ϑ as the least set such that $\rho \models \overline{E} : \vartheta$ and prove that $\vartheta \subseteq \rho(z_\bullet)$; intuitively, if $\overline{E}$ has free variables $z_1, \ldots, z_m$ then ϑ consists of all values $\lfloor \overline{E}[V_1/z_1, \ldots, V_m/z_m] \rfloor$ where $V_i \in \rho(z_\bullet)$. We perform a similar development for $\vartheta_0, \ldots, \vartheta_j$ and this takes care of (a). Next consider $\{V_1, \ldots, V_k\}_{V_0}^{l} \in \vartheta$ and assume that $V_0 \in \vartheta_0$. Since $\vartheta_0 \subseteq \rho(z_\bullet)$, as above, we have $V_0 \in \rho(z_\bullet)$ and by $\mathcal{F}_{RM}^{DY}$ we get $\wedge_{i=j+1}^{k}V_i \in \rho(z_\bullet)$ and $\wedge_{i=j+1}^{k}\neg\mathsf{RM}(V_i, \mathcal{I}(V_i), x_i, \mathcal{N}) \Rightarrow l_\bullet \in \psi$. Since $\lfloor x_i \rfloor = z_\bullet$ this takes care of (b) and (c); furthermore $\overline{P}$ has type $(\{z_\bullet\}, \mathcal{N}_f \cup \{n_\bullet\}, \mathcal{A}_\kappa, \mathcal{A}_{ENC})$ and the induction hypothesis then takes care of (d).

The remaining cases are similar.

For the dynamic property, we say that P_{sys} guarantees *dynamic confidentiality* with respect to the annotations in P_{sys} if the reference monitor RM cannot abort $P_{sys} \mid \overline{Q}$ regardless of the choice of the attacker Q.

Similarly, for static property we say that P_{sys} guarantees *static confidentiality* with respect to the annotations in P_{sys} if there exists ρ, κ, β and ψ such that $\rho, \kappa, \beta, \emptyset \models P$ and $(\rho, \kappa, \beta, \emptyset)$ satisfies $\mathcal{F}_{RM}^{DY}$.

Theorem 6.5 *If P guarantees static confidentiality then P guarantees dynamic confidentiality.*

Proof. If $\rho, \kappa, \beta, \emptyset \models P_{sys}$ and $(\rho, \kappa, \beta, \emptyset)$ satisfies $\mathcal{F}_{RM}^{DY}$ then, by Theorems 6.3 and 6.4, RM does not abort $P_{sys} \mid \overline{Q}$ regardless of the choice of attacker Q.

That is, P ensures within and from confidentiality no matter which attacker it is composed with.

Example 6.3 *Consider the CCI protocol in Example 6.2 with within and from annotations. The annotations formalise the intuition that the value CCI is supposed to be bound to the variable $yCCI$, only.*

$$\begin{array}{l}((\nu\ CCI[\mathsf{within}\ yCCI])\langle A, B, \{CCI\}_K\rangle.0\\ |\\ (A, B; y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{; yCCI[\mathsf{from}\ CCI]\}^l_K\ \mathsf{in}\ 0)\end{array}$$

The analysis finds that the confidentiality property may be violated when the process is put in parallel with the attacker's process, i.e. $P \mid P_\bullet$:

$$\{l, l_\bullet\} \subseteq \psi$$

The first element l reports that variable bindings after the successful decryption of label l is not allowed: the value bound to the variable may come from the attacker. This is because that the attacker knows the key K. Therefore he is able to generate encryptions that are accepted by the principal B. The second element $l_\bullet$ reports that a value may be bound to a variable belonging to the attacker after a successful decryption, i.e. the value becomes known to the attacker. This corresponds to the problem shown in Example 2.2.

The remaining part of the analysis results is as in Example 4.3 except that annotations are recorded in the analysis component β:

$$\begin{array}{l}\{\{\lfloor CCI\rfloor\}_{\lfloor K\rfloor}\} \subseteq \rho(\lfloor y\rfloor)\\ \{\lfloor CCI\rfloor\} \subseteq \rho(\lfloor yCCI\rfloor)\\ \{\langle\lfloor A\rfloor, \lfloor B\rfloor, \{\lfloor CCI\rfloor\}_{\lfloor K\rfloor}\rangle\} \subseteq \kappa\\ \{\lfloor yCCI\rfloor\} \subseteq \beta(\lfloor CCI\rfloor)\\ \{l, l_\bullet\} \subseteq \psi\end{array}$$

6.6 Confidentiality Analysis at the Meta Level

The meta level confidentiality analysis is identical to the original analysis listed in Table 4.2 in Chapter 4 except that the confidentiality analysis of Table 6.2 is

used to analyse object level constructs. Similar to the object level analysis, in Table 6.2, the environment, β, and the error component, ψ, are included in the meta level confidentiality analysis, i. e. the analysis is of the form

$$\rho, \kappa, \beta, \psi \models_\Gamma M$$

Example 6.4 *Consider the object level process from Example 6.3. The meta level process may be modelled as the following, where the* within *and* from *annotations from different sessions are distinguished by the indices. Assume it is deployed in a scenario where there are many principals A_i and many principals B_j and all of them share the same key, K, which is kept secret from the attacker by using the restriction $(\nu\ K)$*

$$\begin{array}{l}
\mathsf{let}\ X \subseteq S_1\ \mathsf{in} \\
\mathsf{let}\ Y \subseteq S_2\ \mathsf{in} \\
(\nu\ K)(\\
\quad |_{i\in X}|_{j\in Y}(\nu\ CCI_{ij}[\mathsf{within}\ yCCI_{ij}])\langle A_i, B_j, \{CCI_{ij}\}_K\rangle.0 \\
| \\
\quad |_{i\in X}|_{j\in Y}(A_i, B_j; y_{ij}).\mathsf{decrypt}\ y_{ij}\ \mathsf{as}\ \{; yCCI_{ij}[\mathsf{from}\ CCI_{ij}]\}_K^{l_{ij}}\ \mathsf{in}\ 0)
\end{array}$$

Note that unlike CCI, the key K does not carry any annotations. This expresses the intuition that K is not allowed to be bound to any variable.

When taking $S_1 = \{1, 2\}$ and $S_2 = \{1, 2\}$, the analysis result of the process gives an non-empty error component:

$$\{l_{11}, l_{12}, l_{21}, l_{22}\} \subseteq \psi$$

The elements in ψ show that variable bindings in the decryptions with label l_{11}, l_{12}, l_{21} and l_{22} violate the within *and* from *annotations. This corresponds to an attack that forces the principal B_j to decrypt a replayed value, which happens because all the principals share the same key K. Consider the following message exchange:*

	A_1	A_2	B_1	*Attacker*
1.1	$\langle A_1, B_1,$ $\{CCI_{11}\}_K\rangle$			$(A_1, B_1; Z_{e1})$
2.1		$\langle A_2, B_2,$ $\{CCI_{22}\}_K\rangle$		$(A_2, B_2; Z_{e2})$
1.2			$(A_1, B_1; y_{11})$ *decrypt* y_{11} *as* $\{; yCCI_{11}[\textit{from } CCI_{11}]\}_K^{l_{11}}$	$\langle A_1, B_1, Z_{e2}\rangle$

In the step 1.2, the following pattern matching can successfully take place in the input construct,

$$(A_1, B_1; y_{11}) \mid \langle A_1, B_1, \{CCI_{22}\}_K\rangle$$

and let the variable y_{11} become bound to the value $\{CCI_{22}\}_K$. This gives rise to the successful decryption

$$\textit{decrypt } y_{11} \textit{ as } \{; yCCI_{11}[\textit{from } CCI_{11}]\}_K^{l_{11}}$$

This decryption results in binding the variable $yCCI_{11}$ to the value CCI_{22}. After the decryption, the principal B_1 then gets hold of the value, CCI_{22}, which is supposed to be accessible by the principal B_2 only. Therefore a confidentiality violation is caused and is recorded by letting l_{11} in ψ.

Example 6.5 *In this example we shall see whether the protocol preserves the confidentiality when deployed in a scenario that each initiator A_i shares a unique key K_{ij} with a responder B_j. The scenario is modelled as the meta level process as,*

$$\begin{array}{l}
\textit{let } X \subseteq S_1 \textit{ in} \\
\textit{let } Y \subseteq S_2 \textit{ in} \\
\quad |_{i\in X}|_{j\in Y}(\nu\ K_{ij})(\\
\quad\quad (\nu\ CCI_{ij}[\textit{within } yCCI_{ij}])\langle A_i, B_j, \{CCI_{ij}\}_{K_{ij}}\rangle.0 \\
\quad\quad | \\
\quad\quad (A_i, B_j; y_{ij}).\textit{decrypt } y_{ij} \textit{ as } \{; yCCI_{ij}[\textit{from } CCI_{ij}]\}_{K_{ij}}^{l_{ij}} \textit{ in } 0)
\end{array}$$

Again the key K_{ij} does not carry any annotation because it is not allow to be bound to any variable.

This time the analysis result gives an empty error component, i.e. $\psi = \emptyset$, which verifies that the protocol holds the confidentiality property.

6.7 Summary

Confidentiality, defined by the International Organization for Standardization (ISO) as "ensuring that information is accessible only to those authorised to have access", is one of the key security properties that cryptographic protocols have to maintain. Many researches in the formal methods field have been carried out for validating confidentiality property of security protocols. Some of the related work includes type system and model checking.

Type systems have been used for checking confidentiality of security protocols. For example, Abadi proposed in [1] a type-based methodology, where each piece of data is annotated as either secret or public according the protocol intention, and confidentiality of a process is guaranteed when a type checker confirms that the process conforms with the annotations by means of applying a set of typing rules.

The model checking approach [6, 5, 18] is another way to verify confidentiality of security protocols. The basic idea is to formalise a model of a system running the protocol and then to use state space exploration to search for attacks. However the search does not always terminate because, in general, the state space for protocols is infinite. Consequently, model checking approaches cannot guarantee the confidentiality property of a protocol, as they can not search through the entire state space.

In the work presented in this chapter, the confidentiality property is clarified by means of the within and from annotations, which basically express where a piece of information is supposed to end up. The control flow analysis is adapted correspondingly to take the annotations into account and searches for violations to the annotations. An empty set of violations then guarantees the absence of confidentiality flaws. Unlike type systems, which only check whether a secret piece of information may be leaked to an attacker, the control flow analysis is able to verify that secret information will never end up in the "wrong hands", i.e. neither an attacker nor a legitimate principal who is not the intended recipient.

Chapter 7

Freshness

Another security property, desirable in some practical applications but not discussed in [1], is key freshness. By this we mean the property that the party to a key establishment process knows that the key is a *new* key. In particular, the party should have evidence that the messages received during the protocol by which the key has been established are *fresh* messages, i.e. they are not replays of *old* messages from a previous session of the protocol.

Example 7.1 *To see why this property is necessary in addition to implicit or explicit key authentication, consider the credit card information protocol from Example 3.1. It is clear that the protocol does not provide key freshness, since B has no way of telling whether the message has just been generated by A, or is a replay of a message sent by A at any time since K was first established. Of course, this lack of key freshness can easily be rectified by including a time stamp or sequence number within the scope of the encrypted message. The absence of key freshness would enable an attacker to force B to accept an old data as a fresh one. In even worse cases, the attacker may fool a principal to re-use an old session key, which might have been compromised, and hence learn some sensitive data encrypted using the old session key. It would therefore seem reasonable to make key freshness a requirement for most applications of key establishment protocols.*

$\mathcal{E} ::=$	*extended terms*
$[n]_s$	name
x	variable
$[m^+]_s$	public key
$[m^-]_s$	private key
$[\{\mathcal{E}_1, \ldots, \mathcal{E}_k\}_{\mathcal{E}_0}]_s$	symmetric encryption
$[\{\!\|\mathcal{E}_1, \ldots, \mathcal{E}_k\|\!\}_{\mathcal{E}_0}]_s$	asymmetric encryption
$\mathcal{P} ::=$	*extended processes*
$\langle \mathcal{E}_1, \ldots, \mathcal{E}_k \rangle.\mathcal{P}$	output
$(\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k).\mathcal{P}$	input
$\mathsf{decrypt}\ \mathcal{E}\ \mathsf{as}\ \{\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k\}^l_{\mathcal{E}_0}\ \mathsf{in}\ \mathcal{P}$	symmetric decryption
$\mathsf{decrypt}\ \mathcal{E}\ \mathsf{as}\ \{\!\|\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k\|\!\}^l_{\mathcal{E}_0}\ \mathsf{in}\ \mathcal{P}$	asymmetric decryption
$(\nu\ [n]_s)\mathcal{P}$	restriction
$(\nu_{\pm}\ [m]_s)\mathcal{P}$	pair restriction
$\mathcal{P}_1 \mid \mathcal{P}_2$	parallel composition
$[!P]_s$	replication
0	nil

Table 7.1: Syntax of LySA calculus extended with session identifiers

7.1 Setting the Scene

Replay attacks are classified, by Syverson in [81], at the highest level as run-external and run-internal attacks, depending on the origin of messages. In this chapter, we restrict our attention to run-external attacks. This type of attacks allows the attacker to obtain messages from one run of a protocol, often referred to as a *session*, and to send them to a principal participating in another run of the protocol. To distinguish messages from different sessions, we extend the LySA calculus with annotations about sessions identifiers. In order to do that, we change the syntax of standard LySA so that each term and process now carry identifiers of the sessions they belong to. In what follows, we assume that SID is a fixed enumerable set of session identifiers s, and denote $\mathcal{E}_1, \mathcal{E}_2, \ldots$ the extended terms and $\mathcal{P}, \mathcal{Q}, \ldots$ the extended processes defined below. Note that variables carry no annotation and therefore we shall consider $[x]_s$ and x to be the same (see below).

We define a function $\mathcal{F}$ and a function $\mathcal{T}$, that map standard terms and processes into the extended ones, by attaching the session identifiers inductively. Note that $\mathcal{F}$ unwinds the syntactic structure of an extended term until reaching a basic term (a name or a variable), while $\mathcal{T}$ unwinds the structure of an extended

process until reaching a nil (which is untagged) or a replication.

Definition 7.1 Distributing Session Identifiers

$$\mathcal{F} : E \times s \to \mathcal{E}$$

$$\mathcal{F}(n, s) = [n]_s$$
$$\mathcal{F}(x, s) = x$$
$$\mathcal{F}(m^+, s) = [m^+]_s$$
$$\mathcal{F}(m^-, s) = [m^-]_s$$
$$\mathcal{F}(\{E_1, \ldots, E_k\}_{E_0}, s) = [\{\mathcal{F}(E_1, s), \ldots, \mathcal{F}(E_k, s)\}_{\mathcal{F}(E_0,s)}]_s$$
$$\mathcal{F}(\{\!|E_1, \ldots, E_k|\!\}_{E_0}, s) = [\{\!|\mathcal{F}(E_1, s), \ldots, \mathcal{F}(E_k, s)|\!\}_{\mathcal{F}(E_0,s)}]_s$$

$$\mathcal{T} : P \times s \to \mathcal{P}$$

$$\mathcal{T}(\langle E_1, \ldots, E_k\rangle.P, s) = \langle \mathcal{F}(E_1, s), \ldots, \mathcal{F}(E_k, s)\rangle.\mathcal{T}(P, s)$$
$$\mathcal{T}((E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P, s) = (\mathcal{F}(E_1, s), \ldots, \mathcal{F}(E_j, s); x_{j+1}, \ldots, x_k).\mathcal{T}(P, s)$$

$$\mathcal{T}(\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}^l_{E_0}\ \mathsf{in}\ P, s) = \mathsf{decrypt}\ \mathcal{F}(E, s)\ \mathsf{as}\ \{\mathcal{F}(E_1, s), \ldots, \mathcal{F}(E_j, s); x_{j+1}, \ldots, x_k\}^l_{\mathcal{F}(E_0,s)}\ \mathsf{in}\ \mathcal{T}(P, s)$$

$$\mathcal{T}(\mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j; x_{j+1}, \ldots, x_k|\!\}^l_{E_0}\ \mathsf{in}\ P, s) = \mathsf{decrypt}\ \mathcal{F}(E, s)\ \mathsf{as}\ \{\!|\mathcal{F}(E_1, s), \ldots, \mathcal{F}(E_j, s); x_{j+1}, \ldots, x_k|\!\}^l_{\mathcal{F}(E_0,s)}\ \mathsf{in}\ \mathcal{T}(P, s)$$

$$\mathcal{T}((\nu\ n)P, s) = (\nu\ [n]_s)\mathcal{T}(P, s)$$
$$\mathcal{T}((\nu_\pm\ m)P, s) = (\nu_\pm\ [m]_s)\mathcal{T}(P, s)$$
$$\mathcal{T}(P \mid Q, s) = \mathcal{T}(P, s) \mid \mathcal{T}(Q, s)$$
$$\mathcal{T}(!P, s) = [!P]_s$$
$$\mathcal{T}(0, s) = 0$$

7.2 Dynamic Property

Technically, the addition of session identifiers to the syntax means that it is necessary to define its semantics as well. By directly using the semantics definition from the Chapter 3 on the extended syntax, the session identifiers will

be carried on to the semantics of values. This extended value domain will be referred to as $FVal$, ranged over by FV built from the grammar

$$\begin{array}{rcl} FV & ::= & [n]_s \\ & | & [m^+]_s \\ & | & [m^-]_s \\ & | & [\{FV_1, \ldots, FV_k\}_{FV_0}]_s \\ & | & [\{\!|FV_1, \ldots, FV_k|\!\}_{FV_0}]_s \end{array}$$

An equivalent relation $FV_1 = FV_2$ is defined to be the least equivalence over $FVal$ that (inductively) ignores the session identifiers. For example, $[n]_s = [n]_{s'}$ for any s and s' and $[\{[n_1]_{s_1}, [n_2]_{s_2}\}_{[n_0]_{s_0}}]_s = [\{[n_1]_{s'_1}, [n_2]_{s'_2}\}_{[n_0]_{s'_0}}]_{s'}$ for any s, s', s_1, s_2, s'_1 and s_2. The semantics of the extended syntax is then defined by the tables for the reduction semantics in Chapter 3 interpreted over extended values $FV \in FVal$.

For the subsequent treatment, it is convenient to introduce an auxiliary operator, $\mathcal{I}$, which extracts the outermost session identifer of an extended value FV, with respect to encryptions, of an extended term.

Definition 7.2 An auxiliary operator

$$\mathcal{I}\colon FV \to SID$$

$$\begin{array}{l} \mathcal{I}([n]_s) = s \\ \mathcal{I}([m^+]_s) = s \\ \mathcal{I}([m^-]_s) = s \\ \mathcal{I}([\{FV_1, \ldots, FV_k\}_{FV_0}]_s) = s \\ \mathcal{I}([\{\!|FV_1, \ldots, FV_k|\!\}_{FV_0}]_s) = s \end{array}$$

The goal of adding session identifers is to distinguish terms from different sessions.

We remove the rule $!P \equiv P \mid !P$ from the structural congruence rules, in Table 3.3, for the LySA calculus, namely $!P$ and $P \mid !P$ are no longer structurally equivalent in this Chapter. On the contrary, the relationship is lifted to the extended processes, which gives the rule (Repl) in Table 7.2.

Following the tradition of the π-calculus, we shall give the extended LySA a reduction semantics. The *reduction relation* $\rightarrow_{\mathcal{R}}$ is the least relation on closed processes that satisfies the rules in Table 3.5. It uses the standard notion of substitution, $\mathcal{P}[FV/x]$, structural congruence, as defined above, and the disciplined treatment of α-conversion.

As far as the semantics is concerned, we consider two variants of *reduction relation* $\rightarrow_{\mathcal{R}}$, graphically identified by a different instantiation of the relation $\mathcal{R}$, which decorates the transition relation. One variant ($\rightarrow_{\mathsf{RM}}$) takes advantage of annotations, the other one ($\rightarrow$) discards them: essentially, the first semantics checks the freshness of messages, while the other one does not:

- the *reference monitor semantics* $\mathcal{P} \rightarrow_{\mathsf{RM}} \mathcal{Q}$ takes

$$\mathbf{RM}(s, s') = (s = s')$$

- the *standard semantics* $\mathcal{P} \rightarrow \mathcal{Q}$ takes, by construction, $\mathcal{R}$ to be universally true.

The rule (Com) expresses that an output $\langle FV_1, \ldots, FV_j, FV_{j+1}, \ldots, FV_k \rangle.\mathcal{P}$ is matched by an input $(FV_1, \ldots, FV_j; x_{j+1}, \ldots, x_k)$ in case the first j elements are pairwise the same when all the session identifiers are recursively removed, by using the equivalent operator $=$. When the matchings are successful each $FV_i (j + 1 \leq i \leq k)$ is bound to the corresponding x_i.

Similarly, the rules (Dec), (ADec) and (ASig) require that in a successful decryption, either symmetric or asymmetric, the first j values FV_i and FV_i' must be equal, and additionally the keys must be either the same, in case of symmetric decryption, or from the same key pair, in case of asymmetric decryption. When the matching is successful, each FV_i is bound to the corresponding x_i. In the *reference monitor semantics* we ensure that the decrypted message comes from the current session by checking whether the first j values FV_i and FV_i' have the same session identifiers, by using the function $\mathcal{I}$.

The rule (Congr) makes use of the function $\mathcal{T}$, which bridges the gap between the semantics defined on the extended processes $\mathcal{P}$ and the structural congruence defined on the standard processes P.

The rules (New) and (ANew) let an extended process evolve inside a restriction, however the restriction operator itself does never disappear.

In case of (Repl), the process is unfolded once. Note that the new session identifier, s', in this case, has to be unique for not mixing the processes up.

The rule (Par) is standard.

Using this semantics, the property of freshness can be defined as follows:

$$\text{(Com)}\ \frac{\wedge_{i=1}^{j} FV_i = FV_i'}{\langle FV_1, \ldots, FV_k\rangle.\mathcal{P} \mid (FV_1', \ldots, FV_j'; x_{j+1}, \ldots, x_k).\mathcal{P}' \rightarrow_{\mathcal{R}} \mathcal{P} \mid \mathcal{P}'[FV_{j+1}'/x_{j+1}, \ldots, FV_k'/x_k]}$$

$$\text{(Dec)}\ \frac{\wedge_{i=0}^{j} FV_i = FV_i' \ \wedge\ \wedge_{i=1}^{j} \mathcal{R}(\mathcal{I}(FV_i), \mathcal{I}(FV_i'))}{\mathsf{decrypt}\ \{FV_1, \ldots, FV_k\}_{FV_0}\ \mathsf{as}\ \{FV_1', \ldots, FV_j'; x_{j+1}, \ldots, x_k\}^{l}_{FV_0'}\ \mathsf{in}\ \mathcal{P} \rightarrow_{\mathcal{R}} \mathcal{P}[FV_{j+1}'/x_{j+1}, \ldots, FV_k'/x_k]}$$

$$\text{(ADec)}\ \frac{\wedge_{i=1}^{j} FV_i = FV_i' \ \wedge\ \wedge_{i=1}^{j} \mathcal{R}(\mathcal{I}(FV_i), \mathcal{I}(FV_i'))}{\mathsf{decrypt}\ \{\!|FV_1, \ldots, FV_k|\!\}_{[m^+]_s}\ \mathsf{as}\ \{\!|FV_1', \ldots, FV_j'; x_{j+1}, \ldots, x_k|\!\}^{l}_{[m^-]_{s'}}\ \mathsf{in}\ \mathcal{P} \rightarrow_{\mathcal{R}} \mathcal{P}[FV_{j+1}'/x_{j+1}, \ldots, FV_k'/x_k]}$$

$$\text{(ASig)}\ \frac{\wedge_{i=1}^{j} FV_i = FV_i' \ \wedge\ \wedge_{i=1}^{j} \mathcal{R}(\mathcal{I}(FV_i), \mathcal{I}(FV_i'))}{\mathsf{decrypt}\ \{\!|FV_1, \ldots, FV_k|\!\}_{[m^-]_s}\ \mathsf{as}\ \{\!|FV_1', \ldots, FV_j'; x_{j+1}, \ldots, x_k|\!\}^{l}_{[m^+]_{s'}}\ \mathsf{in}\ \mathcal{P} \rightarrow_{\mathcal{R}} \mathcal{P}[FV_{j+1}'/x_{j+1}, \ldots, FV_k'/x_k]}$$

$$\text{(New)}\ \frac{\mathcal{P} \rightarrow_{\mathcal{R}} \mathcal{P}'}{(\nu\ [n]_s)\mathcal{P} \rightarrow_{\mathcal{R}} (\nu\ [n]_s)\mathcal{P}'}$$

$$\text{(ANew)}\ \frac{\mathcal{P} \rightarrow_{\mathcal{R}} \mathcal{P}'}{(\nu_{\pm}\ [m]_s)P \rightarrow_{\mathcal{R}} (\nu_{\pm}\ [m]_s)\mathcal{P}'}$$

$$\text{(Par)}\ \frac{\mathcal{P}_1 \rightarrow_{\mathcal{R}} \mathcal{P}_1'}{\mathcal{P}_1 \mid \mathcal{P}_2 \rightarrow_{\mathcal{R}} \mathcal{P}_1' \mid \mathcal{P}_2}$$

$$\text{(Congr)}\ \frac{P \equiv P' \ \wedge\ \mathcal{T}(P', s) \rightarrow_{\mathcal{R}} \mathcal{T}(P'', s)}{\mathcal{T}(P, s) \rightarrow_{\mathcal{R}} \mathcal{T}(P'', s)}$$

$$\text{(Repl)}\ [!P]_s \rightarrow_{\mathcal{R}} \mathcal{T}(P, s) \mid [!P]_{s'}$$

Table 7.2: Operational Semantics $\mathcal{P} \rightarrow_{\mathcal{R}} \mathcal{P}'$, parameterized on $\mathcal{R}$

Definition 7.3 (Freshness) A process $\mathcal{P}$ ensures the *freshness property* if for all the executions

$$\mathcal{P} \rightarrow^* \mathcal{P}' \rightarrow_{\mathcal{R}} \mathcal{P}''$$

such that there does not exist i $(1 \leq i \leq j+1)$: $\mathcal{I}(FV_i) \neq \mathcal{I}(FV_i')$ when $\mathcal{P}' \rightarrow_{\mathcal{R}} \mathcal{P}''$ is derived using (Dec) on

$$\begin{array}{ll} \mathsf{decrypt} & [\{FV_1, \ldots, FV_k\}_{FV_0}]_s \text{ as} \\ & \{FV_1', \ldots, FV_j'; x_{j+1}, \ldots, x_k\}^l_{FV_0'} \text{ in } \mathcal{P} \end{array}$$

or using (ADec) or (ASig) on

$$\begin{array}{ll} \mathsf{decrypt} & [\{\!|FV_1, \ldots, FV_k|\!\}_{[m^+]_{s'}}]_s \text{ as} \\ & \{\!|FV_1', \ldots, FV_j'; x_{j+1}, \ldots, x_k|\!\}^l_{[m^-]_{s'}} \text{ in } \mathcal{P} \end{array}$$

or

$$\begin{array}{ll} \mathsf{decrypt} & \{\!|FV_1, \ldots, FV_k|\!\}_{[m^-]_{s'}} \text{ as} \\ & \{\!|FV_1', \ldots, FV_j'; x_{j+1}, \ldots, x_k|\!\}^l_{[m^+]_{s'}} \text{ in } \mathcal{P} \end{array}$$

It says that an extended process $\mathcal{P}$ ensures freshness property if there is no violation of the annotations in any of its execution.

Next, we shall show how to extend the original analysis from Chapter 3 such that it can be used to check whether a process ensures within and from confidentiality according to Definition 7.3.

7.3 Static Property

To capture violations of freshness annotations, similar to the authentication analysis, we make use of an error component ψ to collect all the labels, which indicate where the violation happens. Formally, we have

$$\psi \subseteq \mathcal{P}(\lfloor Lab \rfloor)$$

A label $l \in \psi$ means that the value binding after a successful decryption, marked with label l, violates the freshness annotations and therefore is not allowed.

The freshness analysis extends the original analysis in a way that the session identifiers are taken into account and at each decryption, the analysis will check whether the freshness property is preserved and any possible violation will be recorded in the error component ψ.

(FN)	$\rho \models [n]_s : \vartheta$	iff $[\lfloor n \rfloor]_s \in \vartheta$
(FNp)	$\rho \models [m^+]_s : \vartheta$	iff $[\lfloor m^+ \rfloor]_s \in \vartheta$
(FNm)	$\rho \models [m^-]_s : \vartheta$	iff $[\lfloor m^- \rfloor]_s \in \vartheta$
(FVar)	$\rho \models x : \vartheta$	iff $\rho(\lfloor x \rfloor) \subseteq \vartheta$
(FE)	$\rho \models [\{\mathcal{E}_1, \ldots, \mathcal{E}_k\}_{\mathcal{E}_0}]_s : \vartheta$	iff $\wedge_{i=0}^{k} \rho \models \mathcal{E}_i : \vartheta_i \wedge$ $\forall FV_0, \ldots, FV_k : \wedge_{i=0}^{k} FV_i \in \vartheta_i \Rightarrow$ $[\{FV_1, \ldots, FV_k\}_{FV_0}]_s \in \vartheta$
(FAE)	$\rho \models [\{\!\lvert \mathcal{E}_1, \ldots, \mathcal{E}_k \rvert\!\}_{\mathcal{E}_0}]_s : \vartheta$	iff $\wedge_{i=0}^{k} \rho \models \mathcal{E}_i : \vartheta_i \wedge$ $\forall FV_0, \ldots, FV_k : \wedge_{i=0}^{k} FV_i \in \vartheta_i \Rightarrow$ $[\{\!\lvert FV_1, \ldots, FV_k \rvert\!\}_{FV_0}]_s \in \vartheta$

Table 7.3: Freshness Analysis of LySA terms, $\rho \models \mathcal{E} : \vartheta$

In the freshness analysis, patter matching ignores the session identifers by using a special set inclusion operator, $\propto$, which is formally defined as,

$FV \propto \vartheta$ if and only if there exists FV' such that $FV = FV'$ and $FV' \in \vartheta$

Note that the equivalent relationship $=$ is defined over $FVal$ that ignores the session identifers.

The rules of analysing expressions (FN), (FNp), (FNm), (FV), (FE) and (FAE) ensure that the session identifers are attached to the values that they are evaluated to.

The rule (FInp) is identical to the original rule (AInp) except for the use of the inclusion operator $\propto$ for pattern matching.

The decryption rules (FDec) and (FADec) also use the operator $\propto$ when there is need to ignore the session identifers. Additionally, they check after a successful decryption whether there is at least one value bound to a variable has the expected session identifer. If a mismatch of session identifer may occur, the corresponding decryption point is recorded in the error component ψ.

The rule (FRep) attaches two different session identifiers to two copies of the process before analysing both of them. The newly generated session identifier, s', has to be unique in order not to mix processes up.

(FOut) $\rho, \kappa, \psi \models \langle \mathcal{E}_1, \ldots, \mathcal{E}_k \rangle.\mathcal{P}$
iff $\wedge_{i=1}^{k} \rho \models \mathcal{E}_i : \vartheta_i \wedge$
$\forall FV_1, \ldots, FV_k \wedge_{i=1}^{k} FV_i \in \vartheta_i \Rightarrow$
$(\langle FV_1, \ldots, FV_k \rangle \in \kappa \ \wedge \ \rho, \kappa, \psi \models \mathcal{P})$

(FInp) $\rho, \kappa, \psi \models (\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k).\mathcal{P}$
iff $\wedge_{i=1}^{j} \rho \models \mathcal{E}_i : \vartheta_i \wedge$
$\forall \langle FV_1, \ldots, FV_k \rangle \in \kappa : \wedge_{i=i}^{j} FV_i \propto \vartheta_i \Rightarrow$
$(\wedge_{i=j+1}^{k} FV_i \in \rho(\lfloor x_i \rfloor) \ \wedge \ \rho, \kappa, \psi \models \mathcal{P})$

(FDec) $\rho, \kappa, \psi \models \mathsf{decrypt}\ \mathcal{E}\ \mathsf{as}\ \{\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k\}^{l}_{\mathcal{E}_0}\ \mathsf{in}\ \mathcal{P}$
iff $\rho \models \mathcal{E} : \vartheta \ \wedge \ \wedge_{i=0}^{j} \rho \models \mathcal{E}_i : \vartheta_i \wedge$
$\forall [\{FV_1, \ldots, FV_k\}_{FV_0}]_s \in \vartheta : \wedge_{i=0}^{j} FV_i \propto \vartheta_i \Rightarrow$
$(\wedge_{i=j+1}^{k} FV_i \in \rho(\lfloor x_i \rfloor) \wedge$
$(\exists i : 1 \leq i \leq j : \mathcal{I}(FV_i) \neq \mathcal{I}(\mathcal{E}_i) \Rightarrow \lfloor l \rfloor \in \psi) \wedge$
$\rho, \kappa, \psi \models \mathcal{P})$

(FADec) $\rho, \kappa, \psi \models \mathsf{decrypt}\ \mathcal{E}\ \mathsf{as}\ \{\!|\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k|\!\}^{l}_{\mathcal{E}_0}\ \mathsf{in}\ \mathcal{P}$
iff $\rho \models \mathcal{E} : \vartheta \ \wedge \ \wedge_{i=0}^{j} \rho \models \mathcal{E}_i : \vartheta_i \wedge$
$\forall [\{\!|FV_1, \ldots, FV_k|\!\}_{FV_0}]_s \in \vartheta : \forall v_0' \propto \vartheta_0 :$
$\forall m^+, m^-, s, s' : \{FV_0, v_0'\} = \{[\lfloor m^+ \rfloor]_s, [\lfloor m^- \rfloor]_{s'}\} \wedge$
$\wedge_{i=1}^{j} FV_i \propto \vartheta_i \Rightarrow$
$(\wedge_{i=j+1}^{k} FV_i \in \rho(\lfloor x_i \rfloor) \wedge$
$(\exists i : 1 \leq i \leq j : \mathcal{I}(FV_i) \neq \mathcal{I}(\mathcal{E}_i) \Rightarrow \lfloor l \rfloor \in \psi) \wedge$
$\rho, \kappa, \psi \models \mathcal{P})$

(FRep) $\rho, \kappa, \psi \models [!P]_s$ iff $\rho, \kappa, \psi \models \mathcal{T}([P]_s) \ \wedge \ \rho, \kappa, \psi \models \mathcal{T}([P]_{s'})$

(FPar) $\rho, \kappa, \psi \models \mathcal{P} \mid \mathcal{Q}$ iff $\rho, \kappa, \psi \models \mathcal{P} \ \wedge \ \rho, \kappa, \psi \models \mathcal{Q}$

(FNew) $\rho, \kappa, \psi \models (\nu[n]_s)\mathcal{P}$ iff $\rho, \kappa, \psi \models \mathcal{P}$

(FNil) $\rho, \kappa, \psi \models 0$ iff $true$

Table 7.4: Freshness Analysis of LySA processes, $\rho, \kappa, \psi \models \mathcal{P}$

The rest of the rules are similar to the original ones in Table 4.1 except that the analysis is now defined for the extended terms and processes.

7.4 Correctness of the Freshness Analysis

We prove below that our analysis respects the operational semantics of extended LySA. More precisely, we prove a subject reduction result for both the standard and the reference monitor semantics: if $\rho, \kappa, \psi \models \mathcal{P}$, then the same triple (ρ, κ, ψ) is a valid estimate for all the states passed through in a computation of $\mathcal{P}$, i.e. for all the derivatives of $\mathcal{P}$. Additionally, we show that when the ψ component is empty, then the reference monitor is useless.

It is convenient to prove the following lemmata. The first states that estimates are resistant to substitution of closed terms for variables, and it holds for both extended terms and processes. The second lemma says that an estimate for an extended processes $\mathcal{P}$ is valid for every process congruent to $\mathcal{P}$, as well.

Lemma 7.4 (Substitution in expressions) *$\rho \models \mathcal{E} : \vartheta$ and $\mathcal{E}' \in \rho(x)$ imply $\rho \models \mathcal{E}[\mathcal{E}'/x] : \vartheta$*

PROOF. The proof proceeds by structural induction over expressions by regarding each of the rules in the analysis.

Case (Name). Assume that $\rho \models [n]_s : \vartheta$. For arbitrary choices of $\mathcal{E}'$ and x it holds that $[n]_s[\mathcal{E}'/x] = [n]_s$ so it is immediate that also $\rho \models [n]_s[\mathcal{E}'/x] : \vartheta$.

Case (Variable). Assume that $\mathcal{E} = x'$, i.e. that $\rho \models x' : \vartheta$ and therefore $\rho(x') \subseteq \vartheta$. Then there are two cases. Either $\mathcal{E} \neq x$ in which case $x'[\mathcal{E}'/x] = x'$ so clearly $\rho \models x'[\mathcal{E}'/x] : \vartheta$. Alternatively, $\mathcal{E} = x$ in which case $x'[\mathcal{E}'/x] = \mathcal{E}'$. Furthermore assume that $\mathcal{E}' \in \rho(x)$ and because $\rho(x') \subseteq \vartheta$, it holds that $\rho \models \mathcal{E}' : \vartheta$ in which case $\rho \models \mathcal{E}[\mathcal{E}'/x] : \vartheta$ by the analysis.

Case (Encryption). Assume that $\mathcal{E} = \{\mathcal{E}_1, \ldots, \mathcal{E}_k\}_{\mathcal{E}_0}$, i.e. $\rho \models \{\mathcal{E}_1, \ldots, \mathcal{E}_k\}_{\mathcal{E}_0} : \vartheta$. The result holds by applying the induction hypothesis on each individual $\mathcal{E}_i$.

Lemma 7.5 (Substitution in processes) *$\rho, \kappa, \psi \models P$ and $\mathcal{E} \in \rho(x)$ imply $\rho, \kappa, \psi \models P[\mathcal{E}/x]$*

PROOF. The proof is done by straightforward induction applying the induction hypothesis on any sub-process and Lemma 7.4 on any sub-terms.

Lemma 7.6 (Congruence) *If* $P \equiv Q$ *and* $\rho, \kappa, \psi \models \mathcal{T}([P]_s)$ *then* $\rho, \kappa, \psi \models \mathcal{T}([Q]_s)$

Proof. The proof amounts to a straightforward inspection of each of the clauses defining $P \equiv Q$, e.g.

In case $P \mid 0 \equiv P$ We assume $\rho, \kappa, \psi \models \mathcal{T}([P \mid 0]_s)$, which amounts to $\rho, \kappa, \psi \models \mathcal{T}([P]_s) \mid 0$ according to the definition of $\mathcal{T}$, then it must be the case that

$$\frac{\rho, \kappa, \psi \models \mathcal{T}([P]_s) \ \wedge \ \rho, \kappa, \psi \models 0}{\rho, \kappa, \psi \models \mathcal{T}([P]_s) \mid 0}$$

by the analysis rule. Therefore $\rho, \kappa, \psi \models \mathcal{T}([P]_s)$ is proved.

Other cases are similar and we skip the proofs because of space.

We are now ready to state the subject reduction result. It expresses that our analysis is semantically correct regardless of the way the semantics is parameterised, furthermore the reference monitor cannot abort $\mathcal{P}$ when ψ is empty.

Theorem 7.7 (Correctness of Freshness Analysis) *if* $\rho, \kappa, \psi \models \mathcal{P}$ *and* $\psi = \emptyset$ *then* $\mathcal{P}$ *ensures freshness.*

Proof. The theorem can be proven by showing an extended subject reduction result that says if $\rho, \kappa, \psi \models \mathcal{P}$ and $\mathcal{P} \rightarrow \mathcal{P}'$ then $\rho, \kappa, \psi \models \mathcal{P}'$, and furthermore if $\psi = \emptyset$ then $\mathcal{P} \rightarrow \mathcal{P}'$ does not violate the freshness property. An induction on the length of the execution sequences then gives that $\mathcal{P}$ ensures freshness for all executions.

In case (Com) we assume

$$\rho, \kappa, \psi \models \langle \mathcal{E}_1, \ldots, \mathcal{E}_k \rangle.\mathcal{P} \mid (\mathcal{E}'_1, \ldots, \mathcal{E}'_j; x_{j+1}, \ldots, x_k).\mathcal{Q}$$

which amounts to:

(a) $\wedge_{i=1}^{k} \rho \models \mathcal{E}_i : \vartheta_i$
(b) $\forall FV_1, \ldots, FV_k : \wedge_{i=1}^{k} FV_i \in \vartheta_i \Rightarrow \langle FV_1, \ldots, FV_k \rangle \in \kappa$
(c) $\rho, \kappa, \psi \models \mathcal{P}$
(d) $\wedge_{i=1}^{j} \rho \models \mathcal{E}'_i : \vartheta'_i$
(e) $\forall \langle FV_1, \ldots, FV_k \rangle \in \kappa : \wedge_{i=1}^{j} FV_i \propto \vartheta'_i \Rightarrow (\wedge_{i=j+1}^{k} FV_i \in \rho(\lfloor x_i \rfloor) \ \wedge \ \rho, \kappa, \psi \models \mathcal{Q})$

Moreover we assume that $\wedge_{i=1}^{j} \mathcal{E}_i = \mathcal{E}'_i$ because
$\langle \mathcal{E}_1, \ldots, \mathcal{E}_k \rangle.\mathcal{P} \mid (\mathcal{E}'_1, \ldots, \mathcal{E}'_j; x_{j+1}, \ldots, x_k).\mathcal{Q} \rightarrow_{\mathcal{R}} \mathcal{P} \mid \mathcal{Q}[\mathcal{E}_{j+1}/x_{j+1}, \ldots, \mathcal{E}_k/x_k]$

and we have to prove $\rho, \kappa, \psi \models \mathcal{P} \mid \mathcal{Q}[\mathcal{E}_{j+1}/x_{j+1}, \ldots, \mathcal{E}_k/x_k]$. From (a) we have $\wedge_{i=1}^{k} \mathcal{E}_i \in \vartheta_i$ since $\wedge_{i=1}^{k} \mathsf{fv}(\mathcal{E}_i) = \emptyset$ and then (b) gives $\langle \mathcal{E}_1, \ldots, \mathcal{E}_k \rangle \in \kappa$. From (d) and the assumption $\wedge_{i=1}^{j} \mathcal{E}_i = \mathcal{E}'_i$ we get $\wedge_{i=1}^{j} \mathcal{E}_i \propto \vartheta'_i$. Now (e) gives $\wedge_{i=j+1}^{k} \mathcal{E}_i \in \rho(\lfloor x_i \rfloor)$ and $\rho, \kappa, \psi \models \mathcal{Q}$. The substitution result (Lemma 7.5) then gives $\rho, \kappa, \psi \models \mathcal{Q}[\mathcal{E}_{j+1}/x_{j+1}, \ldots, \mathcal{E}_k/x_k]$ and together with (c) this gives the required result.

The second part is trivial: when $\psi = \emptyset$, obviously
$\langle \mathcal{E}_1, \ldots, \mathcal{E}_k \rangle.\mathcal{P} \mid (\mathcal{E}'_1, \ldots, \mathcal{E}'_j; x_{j+1}, \ldots, x_k).\mathcal{Q} \rightarrow_{\mathsf{RM}} \mathcal{P} \mid \mathcal{Q}[\mathcal{E}_{j+1}/x_{j+1}, \ldots, \mathcal{E}_k/x_k]$

In case (Decr) we assume

$$\rho, \kappa, \psi \models \mathsf{decrypt}\ [\{\mathcal{E}_1, \ldots, \mathcal{E}_k\}_{\mathcal{E}_0}]_s\ \mathsf{as}\ \{\mathcal{E}'_1, \ldots, \mathcal{E}'_j; x_{j+1}, \ldots, x_k\}^{l}_{\mathcal{E}'_0}\ \mathsf{in}\ \mathcal{P}$$

which amounts to:

(f) $\wedge_{i=0}^{k}\ \rho \models \mathcal{E}_i : \vartheta_i$
(g) $\forall FV_0, \ldots, FV_k : \wedge_{i=0}^{k} FV_i \in \vartheta_i \Rightarrow [\{FV_1, \ldots, FV_k\}_{FV_0}]_s \in \vartheta$
(h) $\wedge_{i=0}^{j} \rho \models \mathcal{E}'_i : \vartheta'_i$
(i) $\forall [\{FV_1, \ldots, FV_k\}_{FV_0}]_{s'} \in \vartheta : \wedge_{i=0}^{j} FV_i \propto \vartheta'_i$
$\Rightarrow (\wedge_{i=j+1}^{k} FV_i \in \rho(\lfloor x_i \rfloor) \wedge$
$(\exists i : 1 \leq i \leq j : \mathcal{I}(FV_i) \neq \mathcal{I}(\mathcal{E}_i) \Rightarrow \lfloor l \rfloor \in \psi) \wedge$
$\rho, \kappa, \psi \models \mathcal{P})$

Furthermore we assume that $\wedge_{i=0}^{j} \mathcal{E}_i \approx \mathcal{E}'_i$ because
$\mathsf{decrypt}\ [\{\mathcal{E}_1, \ldots, \mathcal{E}_k\}_{\mathcal{E}_0}]_s\ \mathsf{as}\ \{\mathcal{E}'_1, \ldots, \mathcal{E}'_j; x_{j+1}, \ldots, x_k\}_{\mathcal{E}'_0}\ \mathsf{in}\ \mathcal{P} \rightarrow_{\mathcal{R}}$
$\mathcal{P}[\mathcal{E}_{j+1}/x_{j+1}, \ldots, \mathcal{E}_k/x_k]$ and we have to prove $\rho, \kappa, \psi \models \mathcal{P}[\mathcal{E}_{j+1}/x_{j+1}, \ldots, \mathcal{E}_k/x_k]$. From (f) and $\wedge_{i=0}^{k} \mathsf{fv}(\mathcal{E}_i) = \emptyset$, we get $\wedge_{i=0}^{k} \mathcal{E}_i \in \vartheta_i$ and then (g) gives $[\{\mathcal{E}_1, \ldots, \mathcal{E}_k\}_{\mathcal{E}_0}]_s \in \vartheta$. From (h) and the assumption $\wedge_{i=0}^{j} \mathcal{E}_i \approx \mathcal{E}'_i$ we get $\wedge_{i=0}^{j} \mathcal{E}_i \propto \vartheta'_i$. Now (i) gives $\wedge_{i=j+1}^{k} \mathcal{E}_i \in \rho(\lfloor x_i \rfloor)$ and $\rho, \kappa, \psi \models \mathcal{P}$. Using Lemma 7.5 we get the required result $\rho, \kappa, \psi \models \mathcal{P}[\mathcal{E}_{j+1}/x_{j+1}, \ldots, \mathcal{E}_k/x_k]$

For the second part of the result we observe that
$\exists i : 1 \leq i \leq j : \mathcal{I}(FV_i) \neq \mathcal{I}(\mathcal{E}_i) \Rightarrow \lfloor l \rfloor \in \psi$ follows from (i) and since $\psi = \emptyset$ it must be the case that $\wedge_{i=1}^{j} RM(\mathcal{I}(FV_i), \mathcal{I}(\mathcal{E}_0))$. Thus the condition of the rule (Decr) are fulfilled for $\rightarrow_{\mathsf{RM}}$.

In case (Repl) we assume
$\rho, \kappa, \psi \models [!P]_s$, which amounts to:

(a) $\rho, \kappa, \psi \models \mathcal{T}([P]_s)$
(b) $\rho, \kappa, \psi \models \mathcal{T}([P]_{s'})$
(c) $\rho, \kappa, \psi \models [!P]_{s'}$ (because ψ does not contain any session information)

(a) together with (c) then gives the required result $\rho, \kappa, \psi \models (\mathcal{T}([P]_s) \mid [!P]_{s'})$.

Furthermore, it is obvious that when $\psi = \emptyset$, $[!P]_s \rightarrow_{\mathsf{RM}} \mathcal{T}([P]_s) \mid [!P]_{s'}$

The cases (Par) and (Res) follow directly from the induction hypothesis. The case (Congr) also uses the congruence result.

7.5 The Attacker

The attacker process in the freshness analysis is constructed in a similar way to the one for the confidentiality analysis in Chapter 6.5. The construction additionally includes letting 1) all the names used by $\mathcal{P}_{sys}$ be in a finite set $\mathcal{N}_{\mathsf{f}}$, 2) all the variables in a finite set $\mathcal{X}_{\mathsf{c}}$, and 3) all the session identifiers in a finite set $\mathcal{S}_{\mathsf{c}}$, and postulating a new extended name $[n_\bullet]_{s_\bullet}$, where $n_\bullet$ is not in $\mathcal{N}_{\mathsf{f}}$, a new session identifier $s_\bullet$ not in $\mathcal{S}_{\mathsf{c}}$, and a new variable $z_\bullet$ not in $\mathcal{X}_{\mathsf{c}}$. This means no overlapping between the names and variables used by the legitimate principals and the ones used by the attacker. Furthermore, a unique label $l_\bullet$ is used to indicate the decryption place of the attacker.

In order to control the number of names and variables used by the attacker, we construct a semantically equivalent process $\overline{\mathcal{Q}'}$, for a process $\mathcal{Q}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$, as follows: 1) all restrictions $(\nu[n]_s)\mathcal{P}$ are $\alpha-$converted into restrictions $(\nu[n']_s)\mathcal{P}$ where n' has the canonical representative $n_\bullet$, 2) all the occurrences of variables x_i in $(\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k).\mathcal{P}$ and *decrypt* $\mathcal{E}$ *as* $\{\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k\}^l_{\mathcal{E}_0}$ *in* $\mathcal{P}$ are $\alpha-$converted to use variables x'_i with canonical representative $z_\bullet$. Therefore $\overline{\mathcal{Q}'}$ only has finitely many canonical names and variables.

Similar to previous, the formula $\mathcal{F}^{DY}_{RM}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$, defined as the conjunction of the five components in Table 7.5, is capable of characterising the potential effect of all attackers $\overline{\mathcal{Q}}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$. The soundness of our Dolev-Yao condition is established by the following Theorem.

Theorem 7.8 (Correctness of the Dolev-Yao Condition) *if* (ρ, κ, ψ) *satisfies* $\mathcal{F}^{DY}_{RM}$ *of type* $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$ *then* $\rho, \kappa, \psi \models \overline{\mathcal{Q}}$ *for all attackers* $\mathcal{Q}$ *of extended type* $(\{z_\bullet\}, \mathcal{N}_{\mathsf{f}} \cup \{[n_\bullet]_{s_\bullet}\}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$*, namely, the canonical variables are in* $\{z_\bullet\}$ *and the canonical names are in* $\mathcal{N}_{\mathsf{f}} \cup \{[n_\bullet]_{s_\bullet}\}$.

Proof. By structural induction on $\overline{\mathcal{Q}}$

(1) $\wedge_{k\in\mathcal{A}_\kappa}\ \forall\, \langle FV_1,\ldots,FV_k\rangle \in \kappa :\ \wedge_{i=1}^{k} FV_i \in \rho(z_\bullet)$
the attacker may learn by eavesdropping

(2) $\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall [\{FV_1,\ldots,FV_k\}_{FV_0}]_s \in \rho(z_\bullet) :$
$FV_0 \propto \rho(z_\bullet) \Rightarrow (\wedge_{i=1}^{k} FV_i \in \rho(z_\bullet) \wedge (\wedge_{i=1}^{k}(\mathcal{I}(FV_i) \neq s_\bullet) \Rightarrow l_\bullet \in \psi))$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall [\{\!|FV_1,\ldots,FV_k|\!\}_{m^+}]_s \in \rho(z_\bullet) :$
$m^- \propto \rho(z_\bullet) \Rightarrow (\wedge_{i=1}^{k} FV_i \in \rho(z_\bullet) \wedge (\wedge_{i=1}^{k}(\mathcal{I}(FV_i) \neq s_\bullet) \Rightarrow l_\bullet \in \psi))$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall [\{\!|FV_1,\ldots,FV_k|\!\}_{m^-}]_s \in \rho(z_\bullet) :$
$m^+ \propto \rho(z_\bullet) \Rightarrow (\wedge_{i=1}^{k} FV_i \in \rho(z_\bullet) \wedge (\wedge_{i=1}^{k}(\mathcal{I}(FV_i) \neq s_\bullet) \Rightarrow l_\bullet \in \psi))$

the attacker may learn by decrypting messages with keys already known

(3) $\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\, FV_0,\ldots,FV_k : \wedge_{i=0}^{k} FV_i \in \rho(z_\bullet) \Rightarrow [\{FV_1,\ldots,FV_k\}_{FV_0}]_{s_\bullet} \in \rho(z_\bullet)$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\, m^+, FV_1,\ldots,FV_k : m^+ \in \rho(z_\bullet) \wedge \wedge_{i=1}^{k} FV_i \in \rho(z_\bullet) \Rightarrow [\{\!|FV_1,\ldots,FV_k|\!\}_{m^+}]_{s_\bullet} \in \rho(z_\bullet)$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\, m^-, FV_1,\ldots,FV_k : m^- \in \rho(z_\bullet) \wedge \wedge_{i=1}^{k} FV_i \in \rho(z_\bullet) \Rightarrow [\{\!|FV_1,\ldots,FV_k|\!\}_{m^-}]_{s_\bullet} \in \rho(z_\bullet)$

the attacker may construct new encryptions using the keys known

(4) $\wedge_{k\in\mathcal{A}_\kappa}\ \forall FV_1,\ldots,FV_k : \wedge_{i=1}^{k} FV_i \in \rho(z_\bullet) \Rightarrow \langle FV_1,\ldots,FV_k\rangle \in \kappa$
the attacker may actively forge new communications

(5) $\{[n_\bullet]_{s_\bullet}\} \cup \mathcal{N}_f \subseteq \rho(z_\bullet)$
the attacker initially has some knowledge

Table 7.5: The Attacker's Capabilities in Freshness Analysis

In case (Dec) we need to find ϑ and $\vartheta_0,\ldots,\vartheta_j$, and show

- (a) $\rho \models \overline{\mathcal{E}} : \vartheta\ \wedge\ \wedge_{i=0}^{j}\rho \models \overline{\mathcal{E}_i} : \vartheta_i$
  and for all $[\{FV_1,\ldots,FV_k\}_{FV_0}]_s \in \vartheta$ with $\wedge_{i=0}^{j} FV_i \propto \vartheta_i$ that:
- (b) $\wedge_{i=j+1}^{k} FV_i \in \rho(x_i)$
- (c) $\wedge_{i=j+1}^{k} \neg RM(\mathcal{I}(\mathcal{E}_i), \mathcal{I}(\mathcal{E}'_i)) \Rightarrow l_\bullet \in \psi$
- (d) $\rho, \kappa, \psi \models \overline{\mathcal{P}}$

We choose ϑ as the least set such that $\rho \models \overline{\mathcal{E}} : \vartheta$ and prove that $\vartheta \subseteq \rho(z_\bullet)$; intuitively, if $\overline{\mathcal{E}}$ has free variables $z_1,\ldots,z_m$ then ϑ consists of value $\overline{\mathcal{E}}[FV_1/z_1,\ldots,FV_m/z_m]$ where $FV_i \in \rho(z_\bullet)$. We perform a similar development for $\vartheta_0,\ldots,\vartheta_j$ and this takes care of (a). Next consider $[\{FV_1,\ldots,FV_k\}_{FV_0}]_s \in \vartheta$ and assume that $FV_0 \propto \vartheta_0$. Since $\vartheta_0 \subseteq \rho(z_\bullet)$, as above, we have $FV_0 \propto \rho(z_\bullet)$ and by $\mathcal{F}_{RM}^{DY}$ we get $\wedge_{i=1}^{k} FV_i \in \rho(z_\bullet)$ and $\neg RM(\mathcal{I}(\mathcal{E}_i), \mathcal{I}(\mathcal{E}'_i)) \Rightarrow l_\bullet \in \psi$. Since $x_i = z_\bullet$ this takes care of (b) and (c); furthermore $\overline{\mathcal{P}}$ has type $(\{z_\bullet\}, \mathcal{N}_{\mathsf{f}} \cup \{[n_\bullet]_{s_\bullet}\}, \mathcal{A}_\kappa, \mathcal{A}_{ENC})$ and

the induction hypothesis then takes care of (d).

In case (Repl) We need to show that

(a) $\rho, \kappa, \psi \models \mathcal{T}([\mathcal{P}]_s)$
(b) $\rho, \kappa, \psi \models \mathcal{T}([\mathcal{P}]_{s'})$

By inspecting the functions $\mathcal{F}$ and $\mathcal{T}$, it is easy to tell that both $\mathcal{T}([\mathcal{P}]_s)$ and $\mathcal{T}([\mathcal{P}]_{s'})$ have the same type as $[!\mathcal{P}]_s$ and thus the induction hypothesis takes care of (a) and (b).

The remaining cases are similar.

The session identifiers in the extended LySA are designed to capture replay attacks and thus ensure the receiving messages are fresh. For the dynamic property, we say that $\mathcal{P}_{sys}$ guarantees *dynamic freshness* with respect to the annotations in $\mathcal{P}_{sys}$ if the reference monitor RM cannot abort $\mathcal{P}_{sys} \mid \overline{\mathcal{Q}}$ regardless of the choice of the attacker $\mathcal{Q}$.

Similarly, for static property we say that $\mathcal{P}_{sys}$ guarantees *static freshness* with respect to the annotations in $\mathcal{P}_{sys}$ if there exists ρ and κ such that $\rho, \kappa \models \mathcal{P} : \emptyset$ and $(\rho, \kappa, \emptyset)$ satisfies $\mathcal{F}^{DY}_{RM}$.

Theorem 7.9 *If $\mathcal{P}$ guarantees static freshness then $\mathcal{P}$ guarantees dynamic freshness.*

PROOF. If $\rho, \kappa \models \mathcal{P}_{sys} : \emptyset$ and $(\rho, \kappa, \emptyset)$ satisfies $\mathcal{F}^{DY}_{RM}$ then, by Theorems 7.7 and 7.8, RM does not abort $\mathcal{P}_{sys} \mid \overline{\mathcal{Q}}$ regardless of the choice of attacker $\mathcal{Q}$.

7.6 Freshness Analysis at the Meta Level

As stated in Chapter 4.2, a meta level process is an extension of the object level process with additional sequence of indices, $i_1, \ldots, i_k$, to names and variables. In this case, an index sequence i_1, i_2 can be viewed as an identifier of the session between the principal A_{i_1} and the principal B_{i_2}.

To show in detail how the meta level freshness analysis works, we use the Wide Mouthed Frog (WMF) protocol as an example. The narration of the Wide Mouthed Frog protocol is as the following (please refer to section 2.4.1 for detail explanations):

$$\begin{array}{lll} 1. & A \rightarrow S: & \{K\}_{K_a} \\ 2. & S \rightarrow B: & \{K\}_{K_b} \end{array}$$

The Wide Mouthed Frog protocol is subject to a replay attack, namely, in the step 2, the principal B may receive a message replayed by the attacker from an old session, $\{K_{old}\}_{K_b}$, and thus accept an old session key K_{old} as a new old. If later on the principal B uses the old session key K_{old} to encrypt a message and in case the key is leaked to an attacker, the attacker is able to perform decryption and get the secret message.

Below is an encoding of the protocol. Note that a sequence of indices, ij, is attached to all the names, variables and encryptions, indicating that they belong to a session between initiator i and responder j.

$$\begin{array}{lll} \mathsf{let}\ X \subseteq S_1\ \mathsf{in}\ \mathsf{let}\ Y \subseteq S_2\ \mathsf{in}\ (\nu\ Ka_{ij})(\nu\ Kb_{ij}) & & \\ \quad |_{i\in X}|_{j\in Y} & !(\nu\ K_{ij})\langle A_{ij}, S_{ij}, [\{K_{ij}\}_{Ka_{ij}}]_{ij}\rangle.0 & \\ \mid \quad |_{i\in X}|_{j\in Y} & !(S_{ij}, B_{ij}; y1_{ij}). & \\ & \mathsf{decrypt}\ y1_{ij}\ \mathsf{as}\ \{; yk_{ij}\}^{l2_{ij}}_{Kb_{ij}}\ \mathsf{in}\ 0 & \\ \mid \quad |_{i\in X}|_{j\in Y} & !(A_{ij}, S_{ij}; z1_{ij}). & \\ & \mathsf{decrypt}\ z1_{ij}\ \mathsf{as}\ \{; zk_{ij}\}^{l1_{ij}}_{Ka_{ij}}\ \mathsf{in} & \\ & \langle S_{ij}, B_{ij}, [\{zk_{ij}\}_{Kb_{ij}}]_{ij}\rangle.0 & \end{array}$$

Taking $S_1 = \{1\}$ and $S_2 = \{1, 2\}$, the analysis holds whenever

$$\{l2_{11}, l2_{12}\} \in \psi$$

Thus the analysis reports possible freshness attacks such that in the decryption $\mathsf{decrypt}\ y1_{ij}\ \mathsf{as}\ \{; yk_{ij}\}^{l2_{ij}}_{Kb_{ij}}\ \mathsf{in}\ 0$, a violation to the freshness property may occur.

One may find an execution that leads to the above violation. This can be attempted for two sessions, one is between A_1 and B_1, and the other is between A_1 and B_2. Consider the following message exchange:

	A	B	S	$Attacker$
1.1	$\langle A_{11}, S_{11}, [\{K_{11}\}_{Ka_{11}}]_{11}\rangle$		$(A_{11}, S_{11}; z1_{11})$ decrypt $z1_{11}$ as $\{; zk_{11}\}^{l1_{11}}_{Ka_{11}}$	
1.2			$\langle S_{11}, B_{11}$ $[\{zk_{11}\}_{Kb_{11}}]_{11}\rangle$	$(S_{11}, B_{11}; ze_{11})$
2.1	$\langle A_{12}, S_{12}$ $[\{K_{12}\}_{Ka_{12}}]_{12}\rangle$		$(A_{12}, S_{12}; z1_{12})$ decrypt $z1_{12}$ as $\{; zk_{12}\}^{l1_{12}}_{Ka_{12}}$	
1.2		$(S_{12}, B_{12}; y1_{12})$ decrypt $y1_{12}$ as $\{; yk_{12}\}^{l2_{12}}_{Kb_{12}}$		$\langle S_{12}, B_{12}, ze_{11}\rangle$

In step 1.1, the server S receives the value $[\{K_{11}\}_{Ka_{11}}]_{11}$ sent from the principal A and generates encryption $[\{K_{11}\}_{Kb_{11}}]_{11}$. This value is, somehow, intercepted by the attacker and becomes bound to the variable ze_{11}. In step 1.2, the attacker pretends to be the server S and sends the encryption $[\{K_{11}\}_{Kb_{11}}]_{11}$ to the principal B. The following pattern matching can successfully take place in the input construct,

$$\langle S_{12}, B_{12}, [\{K_{11}\}_{Kb_{11}}]_{11}\rangle \mid (S_{12}, B_{12}; y1_{12})$$

and let the variable $y1_{12}$ become bound to the value $[\{K_{11}\}_{Kb_{11}}]_{11}$. In the decryption decrypt $y1_{12}$ as $\{; yk_{12}\}^{l2_{12}}_{Kb_{12}}$, a violation to the freshness property then happens: the session identifer of K_{11}, e.g. 11, does not match the session identifer of yk_{12}, e.g. 12. The violation is recorded in the error component as $l2_{12} \in \psi$. The attack corresponds to the replay attack described before.

7.7 Summary

Many times protocols are successfully attacked when an honest principal incorrectly accepts messages, "believing" that they posses some properties (e.g. freshness). Whether a technique detects certain attacks depends upon successfully identifying whether the accepted messages have the properties they claimed (e.g. fresh).

Several papers deal with replay attacks and freshness, i.e. [44, 43, 45] and [22], where the approach is based on type (and effects) systems that statically guarantee entity authentication of protocols. Gordon and Jeffrey [44, 43, 45] defined type (and effects) systems that statically guarantee authentication of protocols specified in a Spi-calculus enriched with assertions *à la* Woo-Lam. In [22], Bugliesi, Focardi, Maffei still use a type and effect system, but use a different technique and a different calculus (the ρ-spi calculus).

In most of the cases, cryptographic protocols use nonce handshakes to establish message freshness, and therefore to prevent replay attacks from happening. Three styles of nonce handshakes may happen [44],

- *Public Out Secret Home (POSH)*: nonces go out in clear and return encrypted
- *Secret Out Public Home (SOPH)*: nonces go out encrypted and return in clear
- *Secret Out Secret Home (SOSH)*: nonces go out encrypted and return encrypted

Note that it is very unlikely a nonce is sent out and received both in clear. With the help of this nonce handshakes classification, as well as the fact that the attacker can manipulate any message in clear, but it has no direct control on the encrypted messages, a conclusion can be drawn safely that it is sufficient only to check whether, after each successful decryption, the decrypted messages in *fresh* or not.

The way to validate freshness, as presented in this chapter, is inspired by the BAN logic [23]. In BAN logic, reasoning about the freshness of an entire message amounts to reasoning about the freshness of its fields, i.e. "if one part of a formula is known to be fresh, then the entire formula must also be fresh".

To summarise, in this chapter we have introduced a sound way to detect replay attacks statically. To do that, we extended the standard LySa calculus with session identifiers and gave it a reduction semantics. The semantics ensures that session identifiers are properly treated along the evolution of a process. On the static side, we extended the control flow analysis [15, 16] to verify the freshness property of the extended processes. The static property ensures that, if the secret information received by a principal is in the right context, then a process is not subject to a run-external attack at execution time. It is also proved that analysing two copies of a process in the framework is sufficient for capturing run-external replay attacks, where similar results can be

found in e.g. Comon & Cortier [63] and Millen [65]. The analysis has been implemented and used to some significant protocols, including classical protocols, e.g. Wide Mouthed Frog, Yahalom, Andrew Secure RPC, Otway-Rees, Needham-Schroeder, Amended Needham-Schroeder. Besides the classical protocols, the analysis has also been applied to other kinds of protocols, like the ones in the family of IEEE 802.16 [52]. The results confirmed that potential replay attacks on the protocols are successfully detected.

Chapter 8

Simple Type Flaws

A *simple type flaw attack* on a security protocol arises when a field, originally intended to have one type, is instead interpreted as having another type. To prevent such attacks, the current techniques [48, 57] consist of systematically associating each message field with a tag representing its intended type. Therefore fields with different types cannot be mixed up. Nevertheless, these may result in requiring extra and somehow unnecessary computational power and network transmission band. This is particularly the case, when resource are limited such as in battery-powered embedded systems like PDAs, cell phones, laptops, etc.

Example 8.1 *Consider a scenario that a principal A sends out an encrypted nonce onto the network and another principal B is expecting an encrypted key received from the network. Assume both encryptions use the same key K, obviously, B could be cheated into accepting the nonce as the key.*

$$\begin{array}{rcrl} A & \rightarrow & : & \{N\}_K \\ & \rightarrow & B: & \{K'\}_K \end{array}$$

In this chapter, we further explore these issues and propose a static analysis technique, based on Control Flow Analysis, for detecting type flaw attacks in the presence of a Dolev-Yao attacker [33]. The proposed approach abstracts the fields of protocol messages to a lower level, such that the misinterpretation can be formally modelled. To this end, we extend the LySa calculus with

special tags, which represent the type of terms. The Control Flow Analysis approximates the behaviour of protocols in terms of the possibly exchanged messages and potential values of variables. The analysis can be working in either a *prescriptive* way, such that type flaws are avoided; or a *descriptive* way, such that type flaws are detected and recorded as violations to the intended types. Furthermore, if no type violation is found, we can prove that the protocol is free of simple type flaw attacks at run time.

8.1 Setting the Scene

Type information is used to identify the intension of each field. To make explicit what type means, we extend the syntax of LySA to cope with types, by using tags to represent the types of terms. The syntax of the LySA is extended to take types into account. A type T is built by the grammar below.

$$\begin{array}{lcl} tag & ::= & agent \\ & | & nonce \\ & | & key \\ & | & enc \\ T & ::= & tag \mid u \end{array}$$

Table 8.1: Syntax of Types

A type T is either a constant type $tag \in Tag$, which is a base type for each term, or a variable type $u \in \mathcal{U}$, which is used to be bound constant types in variable binding.

The syntax of the LySA calculus is extended to include types as listed in Table 8.2, while the rest of the constructs are the same as the original ones in Table 3.1.

In expressions, each variable x is annotated by a type T, i.e. an extended variable may be of the form x^{tag} or x^{u}. This is essential in modelling protocols for different purposes, which shall be explained later.

To describe the type intentions of protocols in LySA, whenever a name n is introduced, i.e. $(\nu\ n : tag)P$ we decorate it with the annotation tag, which specifies the type associated to it. The restriction construct is further overloaded for declaring the expected value of a type variable, for example $(\nu\ u : key)P$ states that the type variable u is supposed to be the type key. At each binding occurrence inside a decryption, we add the annotations T to each variable

$E ::=$	*terms*
x^T	variable
$\vdots$	
$P ::=$	*processes*
$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}\}_{E_0}^l\ \mathsf{in}\ P$	symmetric decryption
$\mathsf{decrypt}\ E\ \mathsf{as}\ \{\!\|E_1, \ldots, E_j; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}\|\!\}_{E_0}^l\ \mathsf{in}\ P$	asymmetric decryption
$(\nu\ n : tag)P$	restriction
$(\nu_{\pm}\ m)P$	pair restriction
$(\nu\ u : tag)P$	type declaration
$\vdots$	

Table 8.2: Syntax of extended LySA calculus with types

x, where T is the expected type of x. As usual we use $l \in Lab$ as a label uniquely identifying each decryption place. Matching patterns, in the form $(E'_1, \ldots, E'_j; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k})$, against tuples of expressions $(E_1, \ldots, E_k)$ in the extended LySA works in a slightly different way from the original one: the list of variables $x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}$ is implicitly partitioned into two groups, the type of which is to be matched, i.e. x^{tag}, and the type of which is to be bound, i.e. x^u, by the kind of the annotated type for each variable. The pattern matching only succeeds when 1) the first j elements are pairwise equal, and 2) the constant types of variables have to be equal to the types of the corresponding expressions. The effects of a successful pattern matching are to bind the remaining expressions E_i to the corresponding variables x_i and also to bind the remaining types of expressions to the corresponding type variables.

Example 8.2 *To exemplify, consider the following two processes.*

$$\begin{aligned} P = &\ (\nu\ N : nonce)\ \mathsf{decrypt}\ \{A, N\}_K\ \mathsf{as}\ \{A; x^{nonce}\}_K^l\ \mathsf{in}\ P' \\ Q = &\ (\nu\ N : nonce)\ \mathsf{decrypt}\ \{A, N\}_K\ \mathsf{as}\ \{A; x^{t}\}_K^l\ \mathsf{in}\ Q' \end{aligned}$$

The pattern matching in the decryption in P succeeds because the type of x, nonce, matches the type of N, and results in binding x to N. The second decryption in Q always succeeds, and results in binding x to N and binding t to nonce.

8.2 Dynamic Property

In the semantics, type declarations are recorded in the abstract environment β, i.e. $\beta : \mathcal{N} \cup \mathcal{U} \rightarrow Tag$ is an abstract environment containing the types of names, $\mathcal{N}$, and the expected types of type variables, $\mathcal{U}$. The operation $\mathcal{I}$, is used to determine the type of each value by looking up in the environment β. The type of each encryption is *enc* implicitly. Recall that values $V \in Val$ are built from the grammar:

$$V ::= n \mid m^+ \mid m^- \mid \{V_1, \ldots, V_k\}_{V_0} \mid \{\!|V_1, \ldots, V_k|\!\}_{V_0}$$

Definition 8.1 Type Determination

- $\mathcal{I} : \quad Val \cup \mathcal{U} \rightarrow Tag$

$$\begin{array}{lcl} \mathcal{I}(n) & = & \beta(n) \\ \mathcal{I}(m^+) & = & \beta(m^+) \\ \mathcal{I}(m^-) & = & \beta(m^-) \\ \mathcal{I}(u) & = & \beta(u) \\ \mathcal{I}(\{V_1, \ldots, V_k\}_{V_0}) & = & enc \\ \mathcal{I}(\{\!|V_1, \ldots, V_k|\!\}_{V_0}) & = & enc \end{array}$$

The type annotations are enforced by a reference monitor, which aborts type mismatched variable bindings. The rules for communication, symmetric and asymmetric restrictions and decryptions are as shown in Table 8.3, the rest of the rules are similar to the ones in the original semantics (Table 3.5) except that the abstract environment β is taken into account, i.e. the operational semantics has the form

$$\beta \vdash P \rightarrow_{\mathcal{R}} P'$$

with the relation $\mathcal{R}$ considered in two variants;

- the standard semantics, $\beta \vdash P \rightarrow P'$, discards the annotations and takes $\mathcal{R}$ to be universally true
- the reference monitor semantics, $\beta \vdash P \rightarrow_{\mathsf{RM}} P'$, takes advantage of the annotations and takes

$$\mathsf{RM}(T_1, T_2) \quad = \quad \begin{cases} T_1 = \mathcal{I}(T_2) & \text{if } T_2 \in \mathcal{U} \\ true & \text{else} \end{cases}$$

  This function affects only type variables, i.e. when T_2 is of the form u. It checks whether the type associated with the type variable u equals to T_1.

Moreover, we define an auxiliary function that matches the constant type of a variable x against the type of a value V. This function affects the variables of constant types.

$$COMP(V, x^T) \quad = \quad \begin{cases} \mathcal{I}(V) = T & \text{if } T \in Tag \\ true & \text{otherwise} \end{cases}$$

The judgement $\beta \vdash P \rightarrow_{\mathcal{R}} P'$ means that the process P can evolve into P', given the type environment β. The extended semantics rules are given in Table 8.3, while the rest of the rules are similar to the ones in Table 3.5.

The rules (Dec), (ADec) and (ASig) in Table 8.3 require that in a successful decryption, either symmetric or asymmetric, besides the first j elements have to be pairwise equal, the values $V_{j+1}, \ldots, V_k$ have to be allowed to be bound to the corresponding variable $x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}$ by the type annotations. More specifically, in case the type of the variable x is a constant type, *tag*, it has to be the same as the type of the corresponding value, in which case the types of the values are bound to the types of the corresponding variable. If it is not the case, a type-mismatching violation is captured by the reference monitor.

The rules (ANew), (AANew) and (ATNew) update β to include the types of the names, the asymmetric key pair and the type variable before proceeding.

Using this semantics, the property of type matching can be defined as follows:

Definition 8.2 (Type matching) A process P ensures *type matching* and therefore is simple type flaw attack free if there are no executions

$$\beta \vdash P \rightarrow^* P' \rightarrow_{\mathcal{R}} P''$$

such that there does not exist i $(j+1 \leq i \leq k)$: $\neg \mathsf{RM}(\mathcal{I}(V_i), T_i)$ when $\beta \vdash P' \rightarrow_{\mathcal{R}} P''$ is derived using (Dec) on

$$\mathsf{decrypt} \quad \{V_1, \ldots, V_k\}_{V_0} \ \mathsf{as} \ \{V_1', \ldots, V_j'; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}\}_{V_0}^{l} \ \mathsf{in} \ P$$

or using (ADec) or (ASig) on

$$\mathsf{decrypt} \quad \{\!|V_1, \ldots, V_k|\!\}_{m^+} \ \mathsf{as} \ \{\!|V_1', \ldots, V_j'; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}|\!\}_{m^-}^{l} \ \mathsf{in} \ P$$

or

$$\mathsf{decrypt} \quad \{\!|V_1, \ldots, V_k|\!\}_{m^-} \ \mathsf{as} \ \{\!|V_1', \ldots, V_j'; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}|\!\}_{m^+}^{l} \ \mathsf{in} \ P$$

(Com) $$\frac{\wedge_{i=1}^{j} V_i = V_i' \wedge \wedge_{i=j+1}^{k} COMP(V_i, x_i^{T_i})}{\beta \vdash \langle V_1, \ldots, V_k \rangle.P \mid (V_1', \ldots, V_j'; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}).P' \rightarrow_{\mathcal{R}} P \mid P'[V_{j+1}'/x_{j+1}^{T_{j+1}}, \ldots, V_k'/x_k^{T_k}]}$$

(Dec) $$\frac{\wedge_{i=0}^{j} V_i = V_i' \wedge \wedge_{i=j+1}^{k} COMP(V_i, x_i^{T_i}) \wedge \wedge_{i=j+1}^{k} \mathcal{R}(\mathcal{I}(V_i), T_i)}{\beta \vdash \mathsf{decrypt}\ \ \{V_1, \ldots, V_k\}_{V_0}\ \mathsf{as}\ \{V_1', \ldots, V_j'; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}\}_{V_0'}^{l}\ \mathsf{in}\ P \rightarrow_{\mathcal{R}} P[V_{j+1}'/x_{j+1}^{T_{j+1}}, \ldots, V_k'/x_k^{T_k}]}$$

(ADec) $$\frac{\wedge_{i=1}^{j} V_i = V_i' \wedge \wedge_{i=j+1}^{k} COMP(V_i, x_i^{T_i}) \wedge \wedge_{i=j+1}^{k} \mathcal{R}(\mathcal{I}(V_i), T_i)}{\beta \vdash \mathsf{decrypt}\ \ \{\!|V_1, \ldots, V_k|\!\}_{m^+}\ \mathsf{as}\ \{\!|V_1', \ldots, V_j'; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}|\!\}_{m^-}^{l}\ \mathsf{in}\ P \rightarrow_{\mathcal{R}} P[V_{j+1}'/x_{j+1}^{T_{j+1}}, \ldots, V_k'/x_k^{T_k}]}$$

(ASig) $$\frac{\wedge_{i=1}^{j} V_i = V_i' \wedge \wedge_{i=j+1}^{k} COMP(V_i, x_i^{T_i}) \wedge \wedge_{i=j+1}^{k} \mathcal{R}(\mathcal{I}(V_i), T_i)}{\beta \vdash \mathsf{decrypt}\ \ \{\!|V_1, \ldots, V_k|\!\}_{m^-}\ \mathsf{as}\ \{\!|V_1', \ldots, V_j'; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}|\!\}_{m^+}^{l}\ \mathsf{in}\ P \rightarrow_{\mathcal{R}} P[V_{j+1}'/x_{j+1}^{T_{j+1}}, \ldots, V_k'/x_k^{T_k}]}$$

(ANew) $$\frac{\beta[n \mapsto tag] \vdash P \rightarrow_{\mathcal{R}} P'}{\beta \vdash (\nu\ n : tag)P \rightarrow_{\mathcal{R}} (\nu\ n : tag)P'}$$

(AANew) $$\frac{\beta[m^+ \mapsto key, m^- \mapsto key] \vdash P \rightarrow_{\mathcal{R}} P'}{\beta \vdash (\nu_{\pm}\ m)P \rightarrow_{\mathcal{R}} (\nu_{\pm}\ m)P'}$$

(ATNew) $$\frac{\beta[u \mapsto tag] \vdash P \rightarrow_{\mathcal{R}} P'}{\beta \vdash (\nu\ u : tag)P \rightarrow_{\mathcal{R}} (\nu\ u : tag)P'}$$

Table 8.3: Operational Semantics $\beta \vdash P \rightarrow_{\mathcal{R}} P'$, parameterized on $\mathcal{R}$. The rest of the rules overlap the original ones as in Table 3.5.

It says that a process P is free of type flaw attacks if there is no violation of the annotations in any of its executions, i.e. each type variable is bound to the expected constant type. Consequently, the reference monitor will never stop any execution step. Note that we only consider the type flaws occurring inside encryptions and decryptions.

Next, we shall show how to extend the original analysis from Chapter 4 such that it can be used to check whether a process ensures type matching according to Definition 8.2.

8.3 Static Property

To capture violations of type annotations, similar to the authentication analysis, we make use of an error component ψ to collect all the error messages, which is a label indicating where the violation happens. Formally,

$$\psi \subseteq \mathcal{P}(\lfloor Lab \rfloor)$$

A label $\lfloor l \rfloor \in \psi$ means that the variable binding inside the decryption, marked with label l, violates the type annotations and therefore is not allowed. Note that the analysis requires that labels are subject to a notion of canonicity. Furthermore, the analysis only analyses a LySA process up to canonical values. The abstract environment β and function $\mathcal{I}$ have to respect this and that is the reason that we overload their domains in the analysis i.e.

$$\begin{array}{ll} \mathcal{I}: & \lfloor Val \rfloor \cup \lfloor \mathcal{U} \rfloor \rightarrow Tag \quad \text{and} \\ \beta: & \lfloor \mathcal{N} \rfloor \cup \lfloor \mathcal{U} \rfloor \rightarrow Tag \end{array}$$

Before commenting on the analysis rules, we introduce two auxiliary functions, each of which generates some logic predicates to be used in the analysis rules.

The first function is a *matching* function. It takes a canonical value U and a variable (with type annotation) as input and matches the type of the variable against the type of the value. This matching only happens when the type of the variable is a constant type, *tag*, and is ignored otherwise.

$$match(U, x^T) \quad = \quad \begin{cases} \forall t : (t = \mathcal{I}(U)) \Rightarrow (t = T) & \text{if } T \in Tag \\ true & \text{otherwise} \end{cases}$$

The second function is a *type-checking* function. It takes a value U and a typed variable as input and checks whether the type of the value U equals to the expected type of the variable. The type-checking happens only when the type of the variable is a variable type u.

$$chk(U, x^T) \quad = \quad \begin{cases} \forall t : (t = \mathcal{I}(U)) \Rightarrow (t = \mathcal{I}(\lfloor T \rfloor)) & \text{if } T \in \mathcal{U} \\ true & \text{otherwise} \end{cases}$$

Example 8.3 *Given an abstract environment β, where $\beta(\lfloor N \rfloor) = nonce$, the two functions, match and chk, generate the following logic predicates for the given arguments,*

$$\begin{cases} match(\lfloor N \rfloor, x^{nonce}) = \forall t : (t = \mathcal{I}(\lfloor N \rfloor)) \Rightarrow (t = nonce) \\ match(\lfloor N \rfloor, x^u) = true \end{cases}$$
$$\begin{cases} chk(\lfloor N \rfloor, x^u) = \forall t : (t = \mathcal{I}(\lfloor N \rfloor)) \Rightarrow (t = \mathcal{I}(\lfloor u \rfloor)) \\ chk(\lfloor N \rfloor, x^{nonce}) = true \end{cases}$$

The type analysis extends the original analysis in a way that the type annotations are taken into account. The extended rules are listed in Table 8.4, while the rest of the rules are similar to the original ones in Table 4.1 except that the abstract environment β is added as an additional analysis component.

The rules for decryptions (SDec) and (SADec) use pattern matching and variable binding as before except that types of the values and the corresponding variables have to be matched as well, using the function *match*. Further the analysis checks whether the bindings are allowed according to the types using the function *chk*, or records the label l in the ψ component.

The rules for restrictions (SNew), (SANew) and (STNew) require that the declared type *tag* for the name n, the asymmetric key pair m^+ and m^- and the type variable u are included in the environment β.

8.4 Correctness of the Simple Type Flaw Analysis

Theorem 8.3 (Correctness of Simple Type Flaw Analysis) *If $\beta \vdash P \rightarrow Q$ and $\rho, \kappa, \beta, \psi \models P$ then also $\rho, \kappa, \beta, \psi \models Q$. Furthermore, if $\psi = \emptyset$ then $P \rightarrow_{RM} Q$*

$$(\text{SVar}) \qquad \rho \models x^T : \vartheta \quad \text{iff } \rho(\lfloor x \rfloor) \subseteq \vartheta$$

$$(\text{SInp}) \qquad \rho, \kappa, \beta, \psi \models (E_1, \ldots, E_j; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}).P$$
$$\text{iff } \wedge_{i=1}^{j}\ \rho \models E_i : \vartheta_i \wedge$$
$$\forall \langle U_1, \ldots, U_k \rangle \in \kappa : \wedge_{i=1}^{j}\ U_i \in \vartheta_i \Rightarrow$$
$$(\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \beta, \psi \models P)$$

$$(\text{SDec}) \quad \rho, \kappa, \beta, \psi \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}\}_{E_0}^{l}\ \mathsf{in}\ P$$
$$\text{iff } \rho \models E : \vartheta\ \wedge\ \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i \wedge$$
$$\forall \{U_1, \ldots, U_k\}_{U_0} \in \vartheta\ \wedge$$
$$\wedge_{i=0}^{j}\ U_i \in \vartheta_i\ \wedge\ \wedge_{i=j+1}^{k} match(U_i, x_i^{T_i}) \Rightarrow$$
$$(\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor)\ \wedge$$
$$(\wedge_{i=j+1}^{k} \neg chk(U_i, x_i^{T_i}) \Rightarrow \lfloor l \rfloor \in \psi) \wedge$$
$$\rho, \kappa, \beta, \psi \models P)$$

$$(\text{SADec}) \quad \rho, \kappa, \beta, \psi \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}|\!\}_{E_0}^{l}\ \mathsf{in}\ P$$
$$\text{iff } \rho \models E : \vartheta\ \wedge\ \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i \wedge$$
$$\forall \{\!|U_1, \ldots, U_k|\!\}_{U_0} \in \vartheta : \forall U_0' \in \vartheta_0 :$$
$$\forall (m^+, m^-) : \{U_0, U_0'\} = \{\lfloor m^+ \rfloor, \lfloor m^- \rfloor\} \wedge$$
$$\wedge_{i=1}^{j}\ U_i \in \vartheta_i\ \wedge\ \wedge_{i=j+1}^{k} match(U_i, x_i^{T_i}) \Rightarrow$$
$$(\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor) \wedge$$
$$(\wedge_{i=j+1}^{k} \neg chk(U_i, x_i^{T_i}) \Rightarrow \lfloor l \rfloor \in \psi)\ \wedge$$
$$\rho, \kappa, \beta, \psi \models P)$$

$$(\text{SNew}) \qquad \rho, \kappa, \beta, \psi \models (\nu\ n : tag)P$$
$$\text{iff } tag \in \beta(\lfloor n \rfloor)\ \wedge\ \rho, \kappa, \beta, \psi \models P$$

$$(\text{SANew}) \quad \rho, \kappa, \beta, \psi \models (\nu_{\pm}\ m)P$$
$$\text{iff } key \in \beta(\lfloor m^+ \rfloor) \wedge key \in \beta(\lfloor m^- \rfloor) \wedge \rho, \kappa, \beta, \psi \models P$$

$$(\text{STNew}) \quad \rho, \kappa, \beta, \psi \models (\nu\ u : tag)P$$
$$\text{iff } tag \in \beta(\lfloor u \rfloor)\ \wedge\ \rho, \kappa, \beta, \psi \models P$$

Table 8.4: Simple Type Analysis of LySA Terms, $\rho \models E : \vartheta$, and Processes, $\rho, \kappa, \beta, \psi \models P$. The remains cases overlap the ones in the original analysis as in Table 4.1 but using the judgement form of the simple type analysis.

PROOF. By induction on the inference of $P \to Q$.

In case (Com) Let we assume

$$\rho, \kappa, \beta, \psi \models \langle V_1, \ldots, V_k \rangle.P \mid (V_1', \ldots, V_j'; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}).Q$$

which amounts to:

(a) $\wedge_{i=1}^{k} \rho \models V_i : \vartheta_i$
(b) $\forall U_1, \ldots, U_k : \wedge_{i=1}^{k} U_i \in \vartheta_i \Rightarrow \langle U_1, \ldots, U_k \rangle \in \kappa$
(c) $\rho, \kappa, \beta, \psi \models P$
(d) $\wedge_{i=1}^{j} \rho \models V_i' : \vartheta_i'$
(e) $\forall \langle U_1, \ldots, U_k \rangle \in \kappa : \wedge_{i=1}^{j} U_i \in \vartheta_i \wedge \wedge_{i=j+1}^{k} match(U_i, x_i^{T_i}) \Rightarrow$
$(\wedge_{i=j+1}^{k} U_i \in \rho(\lfloor x_i \rfloor) \wedge (\wedge_{i=j+i}^{k} \neg chk(U_i, x_i^{T_i}) \Rightarrow \lfloor l \rfloor \in \psi) \wedge \rho, \kappa, \beta, \psi \models Q : \psi)$

Moreover we assume that $\wedge_{i=1}^{j} V_i = V_i'$ and $\wedge_{i=j+1}^{k} COMP(V_i, x_i^{T_i})$ because $\langle V_1, \ldots, V_k \rangle.P \mid (V_1, \ldots, V_j; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}).Q \to P \mid Q[V_{j+1}/x_{j+1}^{T_{j+1}}, \ldots, V_k/x_k^{T_k}]$ and we have to prove $\rho, \kappa, \beta, \psi \models P \mid Q[V_{j+1}/x_{j+1}^{T_{j+1}}, \ldots, V_k/x_k^{T_k}]$. From (a) we have $\wedge_{i=1}^{k} V_i \in \vartheta_i$ since $\wedge_{i=1}^{k} \mathsf{fv}(V_i) = \emptyset$ and then (b) gives $\langle V_1, \ldots, V_k \rangle \in \kappa$.

From (d), the assumption $\wedge_{i=1}^{j} V_i = V_i'$ and $\wedge_{i=j+1}^{k} COMP(V_i, x_i^{T_i})$ we get $\wedge_{i=1}^{j} V_i \in \vartheta_i'$ and $\wedge_{i=j+1}^{k} match(V_i, x_i^{T_i})$. Now (e) gives $\wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor)$ and $\rho, \kappa, \beta, \psi \models Q$. The substitution result in Lemma **??** then gives $\rho, \kappa, \beta, \psi \models Q[V_{j+1}/x_{j+1}^{T_{j+1}}, \ldots, V_k/x_k^{T_k}]$ and together with (c) this gives the required result.

The second part is straightforward: when $\psi = \emptyset$, obviously $\wedge_{i=j+1}^{k} chk(U_i, x_i^{T_i})$, which corresponds to $\wedge_{i=j+1}^{k} RM(U_i, T_i)$. Thus the condition of the rule (Com) are fulfilled for $\to_{RM}$.

In case (Dec) we assume

$$\rho, \kappa, \beta, \psi \models \mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}\ \mathsf{as}\ \{V_1', \ldots, V_j'; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}\}_{V_0'}^{l}\ \mathsf{in}\ P$$

which amounts to:

(f) $\wedge_{i=0}^{k}\ \rho \models V_i : \vartheta_i$
(g) $\forall U_0, \ldots, U_k : \wedge_{i=0}^{k} U_i \in \vartheta_i \Rightarrow \{U_1, \ldots, U_k\}_{U_0} \in \vartheta$
(h) $\wedge_{i=0}^{j} \rho \models V_i' : \vartheta_i' \wedge$
(i) $\forall \{U_1, \ldots, U_k\}_{U_0} \in \vartheta : \wedge_{i=0}^{j} U_i \in \vartheta_i' \wedge \wedge_{i=j+1}^{k} match(U_i, x_i^{T_i}) \Rightarrow$
$(\wedge_{i=j+1}^{k} U_i \in \rho(\lfloor x_i \rfloor) \wedge (\wedge_{i=j+1}^{k} \neg chk(U_i, x_i^{T_i}) \Rightarrow \lfloor l \rfloor \in \psi) \wedge \rho, \kappa, \beta, \psi \models P)$

Furthermore we assume that $\wedge_{i=0}^{j} V_i = V_i'$ and $\wedge_{i=j+1}^{k} COMP(V_i, x_i^{T_i})$ because $\mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}\ \mathsf{as}\ \{V_1', \ldots, V_j'; x_{j+1}^{T_{j+1}}, \ldots, x_k^{T_k}\}_{V_0'}^{l}\ \mathsf{in}\ P \to$

$P[V_{j+1}/x_{j+1}^{T_{j+1}}, \ldots, V_k/x_k^{T_k}]$ and we have to prove
$\rho, \kappa, \beta, \psi \models P[V_{j+1}/x_{j+1}^{T_{j+1}}, \ldots, V_k/x_k^{T_k}]$. From (f) and $\wedge_{i=0}^{k} \mathsf{fv}(V_i) = \emptyset$, we get $\wedge_{i=0}^{k} V_i \in \vartheta_i$ and then (g) gives
$\{V_1, \ldots, V_k\}_{V_0} \in \vartheta$. From (h), the assumption $\wedge_{i=0}^{j} V_i = V_i'$ and
$\wedge_{i=j+1}^{k} COMP(V_i, x_i^{T_i})$ we get $\wedge_{i=0}^{j} V_i \in \vartheta_i'$ and $\wedge_{i=j+1}^{k} match(V_i, x_i^{T_i})$. Now (i) gives $\wedge_{i=j+1}^{k} U_i \in \rho(\lfloor x_i \rfloor)$ and $\wedge_{i=j+1}^{k} \neg chk(U_i, x_i^{T_i}) \Rightarrow l \in \psi$ and $\rho, \kappa, \beta, \psi \models P$. Using Lemma A.2 we get the required result $\rho, \kappa, \beta, \psi \models P[V_{j+1}/x_{j+1}^{T_{j+1}}, \ldots, V_k/x_k^{T_k}]$.

For the second part of the result we observe that
$\wedge_{i=j+1}^{k} chk(U_i, x_i^{T_i}) \Rightarrow \lfloor l \rfloor \in \psi$ follows from (i) and since $\psi = \emptyset$ it must be the case that $\wedge_{i=j+1}^{k} chk(U_i, x_i^{T_i})$ and correspondingly $\wedge_{i=j+1}^{k} RM(U_i, T_i)$. Thus the condition of the rule (Dec) are fulfilled for $\rightarrow_{\mathsf{RM}}$.

In case (ADec) and (ASig) are similar.

In case (ATNew) we assume $\rho, \kappa, \beta, \psi \models (\nu\ u : tag)P$, which amounts to:

(a) $tag \in \beta(\lfloor u \rfloor)$
(b) $\rho, \kappa, \beta, \psi \models P$

Furthermore we assume that $\beta[u \mapsto tag] \vdash P \rightarrow Q$. By applying the induction hypothesis on (b), we have $\rho, \kappa, \beta, \psi \models Q$, which together with (a) gives the expected result that $\rho, \kappa, \beta, \psi \models (\nu\ n : tag)Q$.

In case (ANew) is similar.

In cases (Par) and (Rep) follow directly from the induction hypothesis.

The case (Congr) also uses the congruence result.

8.5 The Attacker

The attacker's capabilities in launching simple type flaw attacks are identical to the ones described in Chapter 4.3. Additionally, the attacker is able to generate new names, $n_\bullet$ and new asymmetric key pairs, $m_\bullet^{\pm}$. We associate them with a special type $t_\bullet \in Tag$, which is within the initial knowledge of the attacker. This association is included in the type environment β such that $\beta(n_\bullet) = t_\bullet$ and $\beta(m_\bullet^{\pm}) = t_\bullet$. Furthermore, we let $\mathcal{N}_f$ contain not only the free names but also

(1) $\wedge_{k\in\mathcal{A}_\kappa}\ \forall\,\langle V_1,\ldots,V_k\rangle\in\kappa:\ \wedge_{i=1}^{k}V_i\in\rho(z_\bullet)$
the attacker may learn by eavesdropping

(2) $\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{V_1,\ldots,V_k\}_{V_0}\in\rho(z_\bullet):V_0\in\rho(z_\bullet)\Rightarrow$
$(\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\wedge(\wedge_{i=1}^{k}\mathcal{I}(V_i)\neq t_\bullet\Rightarrow l_\bullet\in\psi))$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{\!|V_1,\ldots,V_k|\!\}_{m^+}\in\rho(z_\bullet):m^-\in\rho(z_\bullet)\Rightarrow$
$(\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\wedge(\wedge_{i=1}^{k}\mathcal{I}(V_i)\neq t_\bullet\Rightarrow l_\bullet\in\psi))$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\{\!|V_1,\ldots,V_k|\!\}_{m^-}\in\rho(z_\bullet):m^+\in\rho(z_\bullet)\Rightarrow$
$(\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\wedge(\wedge_{i=1}^{k}\mathcal{I}(V_i)\neq t_\bullet\Rightarrow l_\bullet\in\psi))$

the attacker may learn by decrypting messages with keys already known

(3) $\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\,V_0,\ldots,V_k:\wedge_{i=0}^{k}V_i\in\rho(z_\bullet)\Rightarrow\{V_1,\ldots,V_k\}_{V_0}\in\rho(z_\bullet)$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\,m^+,V_1\ldots,V_k:m^+\in\rho(z_\bullet)\wedge\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\Rightarrow$
$\{\!|V_1,\ldots,V_k|\!\}_{m^+}\in\rho(z_\bullet)$

$\wedge_{k\in\mathcal{A}_{\mathsf{Enc}}}\ \forall\,m^-,V_1\ldots,V_k:V_0^-\in\rho(z_\bullet)\wedge\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\Rightarrow$
$\{\!|V_1,\ldots,V_k|\!\}_{m^-}\in\rho(z_\bullet)$

the attacker may construct new encryptions using the keys known

(4) $\wedge_{k\in\mathcal{A}_\kappa}\ \forall V_1,\ldots,V_k:\wedge_{i=1}^{k}V_i\in\rho(z_\bullet)\Rightarrow\langle V_1,\ldots,V_k\rangle\in\kappa$
the attacker may actively forge new communications

(5) $\{n_\bullet,m_\bullet^{\pm},t_\bullet\}\cup\mathcal{N}_f\subseteq\rho(z_\bullet)$
the attacker initially has some knowledge

Table 8.5: The Attacker's Capabilities in Simple Type Flaw Attacks

all the types Tag. A unique label $l_\bullet$ is used to indicate the decryption place of the attacker.

Theorem 8.4 (Correctness of Dolev-Yao Condition) *If (ρ,κ,β,ψ) satisfies $\mathcal{F}_{RM}^{DY}$ of type $(\mathcal{N}_\mathsf{f},\mathcal{A}_\kappa,\mathcal{A}_{\mathsf{Enc}})$ then $\rho,\kappa,\beta,\psi\models\overline{Q}$ for all attackers Q of extended type $(\{z_\bullet\},\mathcal{N}_\mathsf{f}\cup\{n_\bullet\},\mathcal{A}_\kappa,\mathcal{A}_{\mathsf{Enc}})$.*

Proof. By structural induction on $\overline{Q}$.

In case of (Dec), i.e. $\overline{Q}$ is the process of the form

$$\mathsf{decrypt}\ \overline{E}\ \mathsf{as}\ \{\overline{E_1},\ldots,\overline{E_j};x_{j+1}^{T_{j+1}},\ldots,x_k^{T_k}\}_{\overline{E_0}}^{l_\bullet}\ \mathsf{in}\ \overline{P}$$

and we need to find ϑ and $\vartheta_0,\ldots,\vartheta_j$ and show

(a) $\rho \models \overline{E} : \vartheta \ \wedge \ \wedge_{i=0}^{j} \rho \models \overline{E_i} : \vartheta_i$
and for all $\{U_1, \ldots, U_k\}_{U_0}^{l} \in \vartheta$ with $U_0 \in \vartheta_0$ and $\wedge_{i=1}^{j} match(U_i, x_i^{T_i})$ that:
(b) $\wedge_{i=j+1}^{k} U_i \in \rho(\lfloor x_i \rfloor)$
(c) $\wedge_{i=j+1}^{k} \neg chk(U_i, x_i^{T_i}) \Rightarrow l_\bullet \in \psi$
(d) $\rho, \kappa, \beta, \psi \models \overline{P} : \psi$

We choose ϑ as the least set such that $\rho \models \overline{E} : \vartheta$ and prove that $\vartheta \subseteq \rho(z_\bullet)$; intuitively, if $\overline{E}$ has free variables $z_1, \ldots, z_m$ then ϑ consists of all values $\lfloor \overline{E}[V_1/z_1, \ldots, V_m/z_m] \rfloor$ where $V_i \in \rho(z_\bullet)$. We perform a similar development for $\vartheta_0, \ldots, \vartheta_j$ and this takes care of (a). Next consider $\{U_1, \ldots, U_k\}_{U_0}^{l} \in \vartheta$ and assume that $U_0 \in \vartheta_0$ and $\wedge_{i=1}^{j} match(U_i, x_i^{T_i})$. Since $\vartheta_0 \subseteq \rho(z_\bullet)$, as above, we have $U_0 \in \rho(z_\bullet)$ and by $\mathcal{F}_{RM}^{DY}$ we get $\wedge_{i=j+1}^{k} U_i \in \rho(z_\bullet)$ and $\wedge_{i=j+1}^{k} \neg chk(U_i, x_i^{T_i}) \Rightarrow l_\bullet \in \psi$. Since $\lfloor x_i \rfloor = z_\bullet$ this takes care of (b) and (c); furthermore $\overline{P}$ has type $(\{z_\bullet\}, \mathcal{N}_f \cup \{n_\bullet\}, \mathcal{A}_\kappa, \mathcal{A}_{ENC})$ and the induction hypothesis then takes care of (d).

The remaining cases are similar.

For the dynamic property, we say that P_{sys} guarantees *dynamic confidentiality* with respect to the annotations in P_{sys} if the reference monitor RM cannot abort $P_{sys} \mid \overline{Q}$ regardless of the choice of the attacker Q.

Similarly, for static property we say that P_{sys} guarantees *static confidentiality* with respect to the annotations in P_{sys} if there exists ρ, κ, β and ψ such that $\rho, \kappa, \beta, \emptyset \models P$ and $(\rho, \kappa, \beta, \emptyset)$ satisfies $\mathcal{F}_{RM}^{DY}$.

Theorem 8.5 *If P guarantees static simple type flaw free then P guarantees dynamic simple type flaw free.*

Proof. If $\rho, \kappa, \beta, \emptyset \models P_{sys}$ and $(\rho, \kappa, \beta, \emptyset)$ satisfies $\mathcal{F}_{RM}^{DY}$ then, by Theorems 8.3 and 8.4, RM does not abort $P_{sys} \mid \overline{Q}$ regardless of the choice of attacker Q. That is, P ensures free of simple type flaw no matter which attacker it is composed with.

Example 8.4 *Consider the protocol from Example 8.2.*

$$\begin{array}{rcrl} A & \rightarrow & : & \{N\}_K \\ & \rightarrow & B : & \{K'\}_K \end{array}$$

Our control flow analysis can work in two ways depending on how the protocol is modelled: either detecting what B received is a wrong one or preventing B from accepting it.

• *In case the goal is to* ***detect*** *any type flaw attack may happen to the protocol, we can model it as the following,*

$$(\nu\ N : nonce)(\nu\ t : key) \quad (\langle A, \{N\}_K\rangle.0 \\ \mid (A; x_{enc}).\ \ \mathsf{decrypt}\ x_{enc}\ \mathsf{as}\ \{; x^t\}^l_K\ in\ 0)$$

where the type of the encrypted message that B received, i.e. x^t, is declared to be of the type key, by $(\nu\ t : key)$. The analysis then gives rise to the analysis components ρ, κ, β and ψ with the following entries:

$$\langle \lfloor A \rfloor, \{\lfloor N \rfloor\}_{\lfloor K \rfloor}\rangle \in \kappa \qquad (\lfloor N \rfloor, nonce) \in \beta \quad (\lfloor t \rfloor, key) \in \beta$$
$$\{\lfloor N \rfloor\}_{\lfloor K \rfloor} \in \rho(\lfloor x_{enc} \rfloor) \qquad \lfloor N \rfloor \in \rho(\lfloor x \rfloor) \qquad \lfloor l \rfloor \in \psi$$

which show that the attack is captured by $\lfloor l \rfloor \in \psi$.

• *In case one wants to* ***prevent*** *such a type flaw attack from happening, the protocol can be modelled as,*

$$(\nu\ N : nonce) \quad (\langle A, \{N\}_K\rangle.0 \\ \mid (A; x_{enc}).\ \ \mathsf{decrypt}\ x_{enc}\ \mathsf{as}\ \{; x^{key}\}^l_K\ in\ 0)$$

It requires that the message inside the encryption that B got has to be a key. In this case, the analysis result becomes:

$$\langle \lfloor A \rfloor, \{\lfloor N \rfloor\}_{\lfloor K \rfloor}\rangle \in \kappa \quad (\lfloor N \rfloor, nonce) \in \beta \quad \psi = \emptyset$$
$$\{\lfloor N \rfloor\}_{\lfloor K \rfloor} \in \rho(\lfloor x_{enc} \rfloor) \quad \rho(\lfloor x^{key} \rfloor) = \emptyset$$

Now $\rho(x^{key}) = \emptyset$ shows that no value binds to the variable x^{key}, i.e. the type flaw attack is successfully prevented.

8.6 Simple Type Flaw Analysis at the Meta Level

To verify the usefulness of our Control Flow Analysis, a number of experiments have been performed on security protocols from the literature. In this section, we shall show the analysis results of some example protocols, which are subject to type flaw attacks, namely the Woo and Lam protocol, version π_1 and the

Andrew Secure RPC protocol (both the original version and the BAN version with type flaw corrected). The analysis results show that those type flaw attacks are successfully captured. Furthermore, it proves that after BAN's correction, the Andrew Secure RPC protocol does not suffer from type flaw attacks any longer.

The narration of the Andrew Secure RPC protocol is the following (please see Chapter 2.4.1 for a detailed explanations).

$$
\begin{array}{lll}
1. & A \rightarrow B: & A, \{N_A\}_K \\
2. & B \rightarrow A: & \{N_A + 1, N_B\}_K \\
3. & A \rightarrow B: & \{N_B + 1\}_K \\
4. & B \rightarrow A: & \{K', N'_B\}_K
\end{array}
$$

Below is an encoding of the protocol in a scenario where each principal A_i and B_j shares a pair of keys, K_{ij}, respectively. This encoding focuses on *detecting* the simple type flaw attacks, i.e. each variable is associated with a type variable, of which the expected type is declared using a type declaration construct.

$$
\begin{array}{lll}
\mathsf{let}\ X \subseteq S\ \mathsf{in}\ (\nu_{i\in X, j\in X}\ K_{ij} : key) & & \\
\quad |_{i\in X}|_{j\in X} & !(\nu\ Na_{ij} : nonce)(\nu\ tx1_{ij} : enc)(\nu\ xnb_{ij} : nonce) & \\
& (\nu\ tx2_{ij} : enc)(\nu\ txk_{ij} : key)(\nu\ txnb'_{ij} : nonce) & \\
& \langle A_i, B_j, A_i, \{Na_{ij}\}_{K_{ij}}\rangle. & \\
& (B_j, A_i; x1_{ij}^{tx1_{ij}}).\ \mathsf{decrypt}\ x1_{ij}^{tx1_{ij}}\ \mathsf{as}\ \{Na_{ij}+1; xnb_{ij}^{txb_{ij}})\}_{K_{ij}}^{l2_{ij}}\ \mathsf{in} & \\
& \langle A_i, B_j, \{xnb_{ij}^{txnb_{ij}} + 1\}_{K_{ij}}\rangle. & \\
& (B_j, A_i; x2_{ij}^{tx2_{ij}}).\ \mathsf{decrypt}\ x2_{ij}^{tx2_{ij}}\ \mathsf{as}\ \{; xk_{ij}^{txk_{ij}}, xnb_{ij}^{\prime txnb'_{ij}}\}_{K_{ij}}^{l3_{ij}}\ \mathsf{in}\ 0 & \\
| \quad |_{i\in X}|_{j\in X} & !(\nu\ Nb_{ij} : nonce)(\nu\ Nb'_{ij} : nonce)(\nu\ K'_{ij} : key) & \\
& (\nu\ ty1_{ij} : enc)(\nu\ tyna_{ij} : nonce) & \\
& (A_i, B_j, A_i; y1_{ij}^{ty1_{ij}}).\ \mathsf{decrypt}\ y1_{ij}^{ty1_{ij}}\ \mathsf{as}\ \{; yna_{ij}^{tyna_{ij}}\}_{K_{ij}}^{l1_{ij}}\ \mathsf{in} & \\
& \langle B_j, A_i, \{yna_{ij}^{tyna_{ij}} + 1, Nb_{ij}\}_{Kb_j}\rangle. & \\
& (A_i, B_j, \{Nb_{ij} + 1; \}_{K_{ij}};). & \\
& \langle B_j, A_i, \{K'_{ij}, Nb'_{ij}\}_{K_{ij}}\rangle.0 &
\end{array}
$$

Taking $S = \{1\}$, the analysis holds whenever

$$\{l3_{11}\} \subseteq \psi$$

Thus, the analysis reports possible simple type flaw attacks such that in the

decryption $\mathsf{decrypt}\ x2_{ij}^{tx2_{ij}}\ \mathsf{as}\ \{; xk_{ij}^{txk_{ij}}, xnb_{ij}^{\prime txnb'_{ij}}\}_{K_{ij}}^{l3_{ij}}$, a type mismatch may occur. An execution that led to this violation is listed below,

	A	B	$Attacker$
1.	$\langle A_1, B_1, A_1, \{Na_{11}\}_{K_{11}}\rangle$	$(A_1, B_1, A_1; y1_{11}^{ty1_{11}})$ $\mathsf{decrypt}\ y1_{11}^{ty1_{11}}\ \mathsf{as}$ $\{; yna_{11}^{tyna_{11}}\}_{K_{11}}^{l1_{11}}$	
2.	$(B_1, A_1; x1_{11}^{tx1_{11}})$ $\mathsf{decrypt}\ x1_{11}^{tx1_{11}}\ \mathsf{as}$ $\{Na_{11}+1; xnb_{11}^{txnb_{11}}\}_{K_{11}}^{l2_{11}}$	$\langle B_1, A_1,$ $\{yna_{11}^{tyna_{11}}+1, Nb_{11}\}_{K_{11}}\rangle$	$(B_1, A_1; Z_{enc}^{tz_{enc}})$
3.	$\langle A_1, B_1, \{xnb_{11}^{txnb_{11}}+1\}_{K_{11}}\rangle$	$(A_1, B_1, \{Nb_{11}+1;\}_{K_{11}};)$	
4.	$(B_1, A_1; x2_{11}^{tx2_{11}})$ $\mathsf{decrypt}\ x2_{11}^{tx2_{11}}\ \mathsf{as}$ $\{; xk_{11}^{txk_{11}}, xnb_{11}^{\prime txnb'_{11}}\}_{K_{11}}^{l3_{11}}$		$\langle B_1, A_1, Z_{enc}^{tz_{enc}}\rangle$

In step 2, the attacker eavesdrops the message, $\langle B_1, A_1, \{Na_{11}+1, Nb_{11}\}_{K_{11}}\rangle$, sent over the network, and binds the variable Z_{enc} to the value $\{Na_{11}{+}1, Nb_{11}\}_{K_{11}}$. Note that the variable yna_{11} has become bound to the value Na_{11} in step 1. He then replays the message to the principal A_1 in step 4. The following successful input can then take place

$$(B_1, A_1; x2_{11}^{tx2_{11}}) \mid \langle B_1, A_1, \{Na_{11}+1, Nb_{11}\}_{K_{11}}\rangle$$

after which the variable $x2_{11}$ is bound to $\{Na_{11}+1, Nb_{11}\}_{K_{11}}$ and the type variable $tx2_{11}$ is bound to the type constant *enc*. This gives rise to the successful decryption

$$\mathsf{decrypt}\ x2_{11}^{tx2_{11}}\ \mathsf{as}\ \{; xk_{11}^{txk_{11}}, xnb_{11}^{\prime txnb'_{11}}\}_{K_{11}}^{l3_{11}}$$

This decryption results in binding the variable xk_{11} to the value $Na_{11}+1$ and binding the type variable txk_{11} to the type *nonce*, and therefore causes a type mismatching violation, i.e. the type of $Na_{11}+1$ is *nonce* while the expected type of xk_{11} is *key*. This violation is recorded by letting $l3_{11}$ in ψ. Some of the entries of the analysis components ρ and κ that are relevant to the attack are shown below,

$\rho:\ Na_{11} \in \rho(yna_{11})$ $\quad \{Na_{11}+1, Nb_{11}\}_{K_{11}} \in \rho(x2_{11})$

$\quad\ Na_{11}+1 \in \rho(xk_{11})$ $\quad Nb_{11} \in \rho(xnb'_{11})$

$\beta:\ key \in \beta(txk_{11})$

$\kappa:\ \langle B_1, A_1, \{Na_{11}+1, Nb_{11}\}_{K_{11}}\rangle \in \kappa$

It is shown above that how the analysis can be used to *detect* simple type flaw attacks. As one may also interested in using the analysis to *prevent* simple type flaw attacks, we present below the encoding for that purpose.

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{let}\ X \subseteq S\ \mathsf{in}\ (\nu_{i\in X, j\in X}\ K_{ij} : key)} \\
& |_{i\in X}|_{j\in X} & !(\nu\ Na_{ij} : nonce) \\
& & \langle A_i, B_j, A_i, \{Na_{ij}\}_{K_{ij}}\rangle. \\
& & (B_j, A_i; x1_{ij}^{enc}).\ \mathsf{decrypt}\ x1_{ij}^{enc}\ \mathsf{as}\ \{Na_{ij}+1; xnb_{ij}^{nonce})\}_{K_{ij}}^{l2_{ij}}\ \mathsf{in} \\
& & \langle A_i, B_j, \{xnb_{ij}^{nonce}+1\}_{K_{ij}}\rangle. \\
& & (B_j, A_i; x2_{ij}^{enc}).\ \mathsf{decrypt}\ x2_{ij}^{enc}\ \mathsf{as}\ \{; xk_{ij}^{key}, xnb'^{nonce}_{ij}\}_{K_{ij}}^{l3_{ij}}\ \mathsf{in}\ 0 \\
| & |_{i\in X}|_{j\in X} & !(\nu\ Nb_{ij} : nonce)(\nu\ Nb'_{ij} : nonce)(\nu\ K'_{ij} : key) \\
& & (A_i, B_j, A_i; y1_{ij}^{enc}).\ \mathsf{decrypt}\ y1_{ij}^{enc}\ \mathsf{as}\ \{; yna_{ij}^{nonce}\}_{K_{ij}}^{l1_{ij}}\ \mathsf{in} \\
& & \langle B_j, A_i, \{yna_{ij}^{nonce}+1, Nb_{ij}\}_{Kb_j}\rangle. \\
& & (A_i, B_j, \{Nb_{ij}+1; \}_{K_{ij}};). \\
& & \langle B_j, A_i, \{K'_{ij}, Nb'_{ij}\}_{K_{ij}}\rangle.0
\end{array}
$$

In the above, each variable is associated with a type constant, i.e. xk_{ij}^{key} states that the value bound to the variable xk_{ij} has to be of the type key. For a version of the process that has been encoded in this way, the analysis holds for $\psi = \emptyset$ and thereby it guarantees absence of complex type flaw attacks.

8.7 Summary

A type flaw attack happens when a field in a message is interpreted as having a type other than the originally intended one. Type flaw attacks on security protocols have been studied for some years, e.g. [48] also adopted the technique of tagging each message field with intended type, and later on, [57] simplified the tag structure for encryption. However these works aim at preventing type flaw

attacks in the protocol execution stage by attaching some extra bits, representing types, to the messages transmitted over the network, and consequently the size of each message is increased, which results in raising unnecessary burden to the underlying network. Other works on type flaw attacks include applying type and effect system to security protocols, e.g. [45], such that a protocol is free of type flaw attacks if it is type checked. Type Systems are normally prescriptive (i.e. they infer types and impose the well-formedness conditions at the same time), while Control Flow Analysis is normally descriptive (i.e. it merely infers the information and then leaves it to a separate step to actually impose demands on when programs are well-formed). Our approach offers a mix of both ways. Indeed, it can be either *descriptive*, i.e. it describes when the protocol does not respect the typing (via binding of type variables) or *prescriptive*, i.e. some flaws are avoided (via matching of tag terms). Under this regard, launching the tool implementing our analysis can then correspond to a sort of approximate type checking. More specifically, our control flow analysis can be used to 1) detect type flaw attacks: it can be applied in the protocol design stage: once a tagged protocol process is analysed to be free of type flaw attacks, it can be used untagged while still ensures security; or 2) prevent type flaw attacks: the tags work in a way such that fields with different types cannot be mixed up. Therefore, it offers flexibility in satisfying different needs.

Chapter 9

Complex Type Flaws

In the last decades, formal analyses of cryptographic protocols have been widely studied and many formal methods have been put forward.

Usually, protocol specification is given at a very high level of abstraction and several implementation aspects, such as the cryptographic ones, are abstracted away. Despite the abstract working hypotheses adopted, many attacks have been found that are independent of these concrete aspects.

However, there are situations in which this abstract view is not completely satisfactory. While, at a high level, messages consists of fields representing certain information, such as the name of a principal, a nonce or a key. This structure can be easily modelled by a process calculus.

Nevertheless, at a more concrete level, a message is nothing but a raw string of bits. In this view, the recipient of a message has to decide the interpretation of the bit string, i.e. how to decompose the string into substrings to be associated to the expected fields (of the expected length) of the message. The message comes with no indication on its arity or on the types of its components. This source of ambiguity can be exploited by an intruder that can fool the recipient into accepting as valid a message different from the expected one. A *complex type flaw attack* arises in this case. Simply speaking, complex type flaw attacks occurs when a concatenation of fields is confused with a single field [79].

Example 9.1 *Consider a scenario that a principal A encrypts its own identity A and a nonce N and sends them out onto the network. Another principal B is expecting an encrypted key received from the network. Assume both encryptions use the same key K, obviously, B could be cheated into accepting the identity and the nonce as the key.*

$$\begin{array}{rcll} A & \rightarrow & : & \{A, N\}_K \\ & \rightarrow & B: & \{K'\}_K \end{array}$$

The central idea of this kind of attacks is to fool a principal by misinterpreting the fields and accepting a concatenation of fields as one single field, as shown in Figure 9.1. However, most existing analyses of certain protocols do not take this kind of attack into account as simply assuming that the number of the fields is known beforehand.

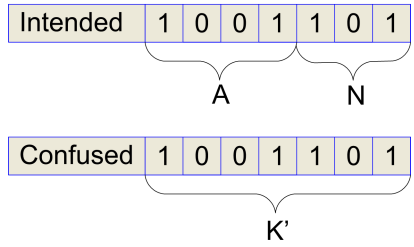


Figure 9.1: Graphical Representation of the Complex Type Flaw Attack

In this chapter complex type flaw attacks are explored and a formal method is proposed for detecting complex type flaw attacks in cryptographic protocols. The proposed approach abstracts the fields of protocol messages to a proper level such that the misinterpretation can be formally modelled. Furthermore, we extend our control flow analysis by multi-to-one variable binding, which models the fields misinterpretation.

9.1 Setting the Scene

A complex type flaw attack happens when there is a possibility that a principal accepts a concatenation of fields as one field. In the LySa calculus, values are passed around among processes by means of pattern matching and variable binding, which models exactly the process that principals acquire knowledge by reading from the network (or performing decryptions). For example, matching patterns of the form $(V_1, \ldots, V_j; x_{j+1}, \ldots, x_k)$ against expressions of the

form $(V'_1, \ldots, V'_k)$ succeeds when the first j values match each other and results in binding the rest of the values $V'_{j+1}, \ldots, V'_k$ to the corresponding variables $x_{j+1}, \ldots, x_k$. An implicit pre-requirement for pattern matching is that patterns and expressions have to be of the same length, which is k is the above example. This requirement ensures that the processes only receive (or decrypt) the messages of which the length is exactly as expected. However, on the other hand, it implicitly removes the possibility of modelling complex type flaws.

Example 9.2 *Consider the complex type flaw attack on the protocol from Example 9.1, with a* LySA *encoding as the following,*

$$\begin{array}{ll} & (\nu\ N)\langle A, \{A, N\}_K\rangle.0 \\ | & (A; x_{enc})^{l_1}.\mathsf{decrypt}\ x_{enc}\ \mathsf{as}\ \{; x_k\}^{l_2}_K\ \mathsf{in}\ 0 \end{array}$$

The principal B is expecting $\{K'\}_K$ but is fooled by accepting $\{A, N\}_K$ and taking (A, N) as the key K' after the decryption. In LySA*, A's movement can be roughly expressed as $(\nu\ N)\langle A, \{A, N\}_K\rangle.0$. Because of the length requirement, x_k can only be binding to a single value but not a concatenation of values, say (A, N).*

To enable LySA to model complex type flaws, we extend the notation of pattern matching and variable binding in a way that the length requirement is relaxed and variables are allowed to be bound to a concatenation of values, i.e. patterns of the form $(V_1, \ldots, V_j; x_{j+1}, \ldots, x_t)$ are allowed to be matched against expressions of the form $(V'_1, \ldots, V'_k)$ when expressions have at least the same number of elements as patterns, i.e. $k \geq t$. Pattern matching succeeds when, as usual, the first j elements are matched and the result is binding values to variables. Since there are more (or the same) values than variables, we partition the values into a number of groups such that each group of values is bound to the corresponding variable. Additionally, since a variable is allowed to be bound to a list of values, there raises the possibility that the list of values is used to encrypt or decrypt a message. These facts require extension to the value domain, which shall be referred to as $PVal$, and is ranged over by PV built from the grammar

$$\begin{array}{rcl} PV & ::= & n \\ & | & m^+ \\ & | & m^- \\ & | & \{PV_1, \ldots, PV_k\}_{(PV'_1, \ldots, PV'_t)} \\ & | & \{\!|PV_1, \ldots, PV_k|\!\}_{(PV'_1, \ldots, PV'_t)} \end{array}$$

For easy presentation, we define a syntax category, $\widetilde{PV}$, which is a list of values.

$$\widetilde{PV} \quad ::= \quad (PV_1, \ldots, PV_k)$$

In order to formally define the extended notation of pattern matching and variable binding, it is handy to introduce three auxiliary functions. The first one is the concatenate operator $\biguplus$, which returns the concatenation of lists of values.

Definition 9.1 $\biguplus$ is the concatenation of lists of values, formally,

$$\begin{array}{l}\biguplus((PV_{11}, \ldots, PV_{1m}), \ldots, (PV_{j1}, \ldots, PV_{jn})) = \\ \quad (PV_{11}, \ldots, PV_{1m}, \ldots, PV_{j1}, \ldots, PV_{jn})\end{array}$$

The second one is to calculate the length of a list.

Definition 9.2

$$Len(PV_1, \ldots, PV_k) = k$$

In order to perform multi-to-one bindings, we resort to a partition operator, $\prod_k$, that given a list of values $(PV_1, \ldots, PV_n)$ returns all the possible partitions composed by k non-empty groups (or lists), where the order among the values is preserved. The function is formally defined as the following,

Definition 9.3 $\prod_k$ partitions a list of values into k non-empty groups, formally,

$$\begin{array}{l}\prod_k(PV_1, \ldots, PV_n) = \\ \left\{\begin{array}{ll}\{(\widetilde{PV'_1}, \ldots, \widetilde{PV'_k}) \mid \\ \quad \biguplus(\widetilde{PV'_1}, \ldots, \widetilde{PV'_k}) = (PV_1, \ldots, PV_n) \wedge \forall 1 \leq i \leq k : \widetilde{PV'_i} \neq [\,]\} & \text{if } n \geq k \\ undefined & \text{otherwise}\end{array}\right.\end{array}$$

Example 9.3 $\prod_2(n_1, n_2, n_3)$ *returns two possible partitions:* $((n_1, n_2), (n_3))$ *and* $((n_1), (n_2, n_3))$

Now binding variables $(x_{j+1}, \ldots, x_t)$ to values $(PV_{j+1}, \ldots, PV_k)$ amounts to partitioning the values into $t - j$, i.e. the number of variables, lists of values and binding variables to the corresponding lists of values $(\widetilde{PV_{j+1}}, \ldots, \widetilde{PV_t}) \in \prod_{t-j}(PV_{j+1}, \ldots, PV_k)$.

Example 9.4 *Pattern matching* $(m; x_1, x_2)$ *against* (m, n_1, n_2, n_3) *succeeds and results in two possible effects,*

- *binding variable* x_1 *to the list of values* (n_1) *and binding variable* x_2 *to* (n_2, n_3)*, or*
- *binding variable* x_1 *to the list of values* (n_1, n_2) *and binding variable* x_2 *to* (n_3)

9.2 Dynamic Property

Complex type flaw attacks may happen when a concatenation of fields is wrongly accepted as one single field. Consequently, at run time, the complex type flaws are checked by a reference monitor, which aborts when there is a possibility that a concatenation of values is bound to a single variable. The semantics has the form

$$P \rightarrow_{\mathcal{R}} P'$$

with the relation $\mathcal{R}$ is considered in two variants;

- the standard semantics, $P \rightarrow P'$ takes $\mathcal{R}$ to be universally true
- the reference monitor semantics, $P \rightarrow_{\mathsf{RM}} P'$, takes

$$\mathsf{RM}(t, k) = (k = t)$$

The reference monitor makes use of the extended notation of pattern matching and variable binding and checks after each successful communication and decryption that whether the pattern, $PV_1, \ldots, PV_j; x_{j+1}, \ldots, x_t$, and the value to be matched against, $PV'_1, \ldots, PV'_k$, have the same length. In case of $k > t$, it means that there exists variable x_s $(j + 1 \leq s \leq t)$, which binds to a concatenation of at least of two values.

The rules for communication, symmetric and asymmetric decryptions are as shown in Table 9.1, the rest of the rules are similar to the ones in the original semantics (Table 3.5).

The semantics rules incorporate a *flatten* function, fl, which flattens the structure of a list by removing the outmost brackets. Formally, it is defined as the following,

$$\text{(Com)}\quad \frac{\wedge_{i=1}^{j} PV_i = PV_i' \ \wedge\ \mathcal{R}(t,k)}{\begin{array}{l}\langle PV_1,\ldots,PV_k\rangle.P \mid (PV_1',\ldots,PV_j';x_{j+1},\ldots,x_t)^l.P' \\ \rightarrow_{\mathcal{R}} P \mid P'[fl(\widetilde{PV}_{j+1})/x_{j+1},\ldots,fl(\widetilde{PV}_t)/x_t] \\ \text{where } (\widetilde{PV}_{j+1},\ldots,\widetilde{PV}_t) \in \prod_{t-j}(PV_{j+1},\ldots,PV_k)\end{array}}$$

$$\text{(Dec)}\quad \frac{\wedge_{i=1}^{j} PV_i = PV_i' \ \wedge\ \widetilde{PV}_0 = \widetilde{PV}_0' \ \wedge\ \mathcal{R}(t,k)}{\begin{array}{l}\mathsf{decrypt}\ \{PV_1,\ldots,PV_k\}_{\widetilde{PV}_0}\ \mathsf{as}\ \{PV_1',\ldots,PV_j';x_{j+1},\ldots,x_t\}^l_{\widetilde{PV}_0'}\ \mathsf{in}\ P \\ \rightarrow_{\mathcal{R}} P[fl(\widetilde{PV}_{j+1})/x_{j+1},\ldots,fl(\widetilde{PV}_t)/x_t] \\ \text{where } (\widetilde{PV}_{j+1},\ldots,\widetilde{PV}_t) \in \prod_{t-j}(PV_{j+1},\ldots,PV_k)\end{array}}$$

$$\text{(ADec)}\quad \frac{\wedge_{i=1}^{j} PV_i = PV_i' \ \wedge\ \mathcal{R}(t,k)}{\begin{array}{l}\mathsf{decrypt}\ \{\!|PV_1,\ldots,PV_k|\!\}_{m^+}\ \mathsf{as}\ \{\!|PV_1',\ldots,PV_j';x_{j+1},\ldots,x_t|\!\}^l_{m^-}\ \mathsf{in}\ P \\ \rightarrow_{\mathcal{R}} P[fl(\widetilde{PV}_{j+1})/x_{j+1},\ldots,fl(\widetilde{PV}_t)/x_t] \\ \text{where } (\widetilde{PV}_{j+1},\ldots,\widetilde{PV}_t) \in \prod_{t-j}(PV_{j+1},\ldots,PV_k)\end{array}}$$

$$\text{(ASig)}\quad \frac{\wedge_{i=1}^{j} PV_i = PV_i' \ \wedge\ \mathcal{R}(t,k)}{\begin{array}{l}\mathsf{decrypt}\ \{\!|PV_1,\ldots,PV_k|\!\}_{m^-}\ \mathsf{as}\ \{\!|PV_1',\ldots,PV_j';x_{j+1},\ldots,x_t|\!\}^l_{m^+}\ \mathsf{in}\ P \\ \rightarrow_{\mathcal{R}} P[fl(\widetilde{PV}_{j+1})/x_{j+1},\ldots,fl(\widetilde{PV}_t)/x_t] \\ \text{where } (\widetilde{PV}_{j+1},\ldots,\widetilde{PV}_t) \in \prod_{t-j}(PV_{j+1},\ldots,PV_k)\end{array}}$$

Table 9.1: Operational Semantics for Complex Type Flaws; $P \rightarrow_{\mathcal{R}} P'$, parametersed on $\mathcal{R}$.

Definition 9.4

$$fl((PV_1,\ldots,PV_k)) = PV_1,\ldots,PV_k$$

The flatten function is applied to a list of values before it is used to replace a variable, by means of the substitution function, e.g. $P[\widetilde{PV}/x]$, in a process. The idea is to avoid introducing structures into the messages along the evolution of a process.

The rule (Com) in Table 9.1 states that in order for a communication to happen, the first j elements in the output and input constructs have to be pairwise equal before variable binding can take place. If this is the case, the rest of the values in the output, $PV_{j+1},\ldots,PV_k$, are partitioned into $t-j$ non-empty lists such that each variable is binding to the corresponding *flattened* value lists. This ensures that in the continuation process, all the messages are plain-structured.

Furthermore, the reference monitor checks the possibility of multi-to-one binding and aborts the execution when it is the case.

The rules (Dec), (ADec) and (ASig) require that in a successful decryption, either symmetric or asymmetric, both the keys and the first j values have to be pairwise equal. In this case the rest of the values are partitioned to some non-empty lists, of which the number equals to the number of the variables, and the lists of values are bound to the corresponding variables. Furthermore, the reference monitor checks the existence of multiple values bound to single variable and aborts the execution in this case.

Example 9.5 *Given the semantics in Table 9.1, the process below evolves as,*

$$\begin{array}{ll} & \langle A, \{n_1, n_2\}_k \rangle.0 \mid (A; x).\mathit{decrypt}\ x\ \mathit{as}\ \{; y\}^l_k\ \mathit{in}\ \langle y \rangle.0 \\ \rightarrow & \mathit{decrypt}\ x\ \mathit{as}\ \{; y\}^l_k\ \mathit{in}\ \langle y \rangle.0[\{n_1, n_2\}_k / x] \\ = & \mathit{decrypt}\ \{n_1, n_2\}_k\ \mathit{as}\ \{; y\}^l_k\ \mathit{in}\ \langle y \rangle.0 \\ \rightarrow & \langle y \rangle.0[n_1, n_2 / y] \\ = & \langle n_1, n_2 \rangle.0 \end{array}$$

Notice that, in the last step, the variable y is substituted by the flatten value list n_1, n_2, which gives a plain-structured output message $\langle n_1, n_2 \rangle$.

Using this semantics, detecting complex type flaw attacks can be defined as follows:

Definition 9.5 (Complex type flaws) A process P is free of complex type flaws if there are no executions

$$P \rightarrow^* P' \rightarrow_{\mathcal{R}} P''$$

such that $t < k$ when $P' \rightarrow_{\mathcal{R}} P''$ is derived using (Com) on

$$\langle PV_1, \ldots, PV_k \rangle.Q \mid (PV'_1, \ldots, PV'_j; x_{j+1}, \ldots, x_t)^l.Q'$$

or using (Dec) on

$$\mathsf{decrypt}\quad \{PV_1, \ldots, PV_k\}_{\widetilde{PV}_0}\ \mathsf{as}\ \{PV'_1, \ldots, PV'_j; x_{j+1}, \ldots, x_t\}^l_{\widetilde{PV'}_0}\ \mathsf{in}\ P$$

or using (ADec) on

$$\mathsf{decrypt}\quad \{\!|PV_1, \ldots, PV_k|\!\}_{m^+}\ \mathsf{as}\ \{\!|PV'_1, \ldots, PV'_j; x_{j+1}, \ldots, x_t|\!\}^l_{m^-}\ \mathsf{in}\ P$$

or using (ASig) on

$$\mathsf{decrypt}\quad \{\!|PV_1, \ldots, PV_k|\!\}_{m^-}\ \mathsf{as}\ \{\!|PV'_1, \ldots, PV'_j; x_{j+1}, \ldots, x_t|\!\}^l_{m^+}\ \mathsf{in}\ P$$

It says that a process P is free of complex type flaw attacks if there is no violation of single-value-to-single-variable-binding in any of its executions, i.e. each variable is bound to a single value. Consequently, the reference monitor will never stop any execution step.

Next, we shall show how to extend the original analysis from Chapter 4 such that it can be used to check whether a process ensures type matching according to Definition 9.5.

9.3 Static Property

In this section we shall describe our control flow analysis. It is motivated by the fact that principals have limited capacities for determining the number of fields in a message he received and a complex type flaw attacks may happen if a principal accepts a concatenation of fields as a single one.

In the LySA calculus, a principal receiving a message is modelled by pattern matching: if the first part (separated by semicolon) of the message matches, the rest of the values will be bound to corresponding values, amounting to a one-to-one binding. In practice, however, principals may be fooled by taking a number of fields as a single one. For modelling this scenario, in the LySA calculus, a multiple-to-one binding is allowed. The control flow analysis then checks whether there is any multiple-to-one binding possibly occurring inside an input or a decryption and records it as a binding violation in an error component ψ. Formally, we have

$$\psi \subseteq \mathcal{P}(\lfloor Lab \rfloor)$$

A label $l \in \psi$ means that a value binding inside the input or decryption, marked with label l, may be a binding violation and therefore is recorded.

Furthermore, we re-define the analysis component ρ as

$$\rho : \lfloor Var \rfloor \rightarrow \mathcal{P}(\lfloor PVal \rfloor^*)$$

to record all the potential value bindings to variables.

The complex type flaw analysis extends the original analysis in a way that multiple-to-one value bindings are allowed and are regarded as violations. Some of the rules are listed in Table 9.3 and Table 9.3, while the rest of them overlap the original ones in Table 4.1.

(TN)	$\rho \models n : \vartheta$	iff $(\lfloor n \rfloor) \in \vartheta$
(TNp)	$\rho \models m^+ : \vartheta$	iff $(\lfloor m^+ \rfloor) \in \vartheta$
(TNm)	$\rho \models m^- : \vartheta$	iff $(\lfloor m^- \rfloor) \in \vartheta$
(TVar)	$\rho \models x : \vartheta$	iff $\rho(\lfloor x \rfloor) \subseteq \vartheta$
(TEnc)	$\rho \models \{E_1, \ldots, E_k\}_{E_0} : \vartheta$	iff $\wedge_{i=0}^{k}\ \rho \models E_i : \vartheta_i \wedge$ $\forall \widetilde{PV}_0, \ldots, \widetilde{PV}_k : \wedge_{i=0}^{k}\ \widetilde{PV}_i \in \vartheta_i \Rightarrow$ $(\{fl(\biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_k))\}_{\widetilde{PV}_0}) \in \vartheta$
(TAEnc)	$\rho \models \{\!\lvert E_1, \ldots, E_k \rvert\!\}_{E_0} : \vartheta$	iff $\wedge_{i=0}^{k}\ \rho \models E_i : \vartheta_i \wedge$ $\forall \widetilde{PV}_0, \ldots, \widetilde{PV}_k : \wedge_{i=0}^{k}\ \widetilde{PV}_i \in \vartheta_i \Rightarrow$ $(\{\!\lvert fl(\biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_k)) \rvert\!\}_{\widetilde{PV}_0}) \in \vartheta$

Table 9.2: Complex Type Flaw Analysis of LySA Terms; $\rho \models E : \vartheta$

The rules for names and asymmetric key pairs (TN), (TNp) and (TNm) state that they are evaluated to lists, of which the elements are their canonical representatives.

The rules for encryptions (TEnc) and (TAEnc) are identical to the original ones except that, since each sub-expression is evaluated to a list of values, the rules compute all the encrypted values that can be formed by concatenating the lists of values that the sub-expressions may be evaluated to, and require that those concatenations are in ϑ.

The rule (TOut), similarly, analyses the output construct and does two things: first all the expressions are evaluated to lists of values and then it is required that all the concatenations of the lists of values found by the evaluation are recorded in κ before the continuation process being analysed.

The rule (TInp) basically looks up in κ for matched tuples and performs variable binding before analysing the continuation process. This is done in the following steps: (1) evaluates the first j expressions and because the results are lists of values, $\widetilde{PV}$, they have to be concatenated to be a single list in order for pattern matching to be performed, (2) looks up in κ for the list of values matching the value list from the previous step, (3) partitions the values into variable-number of groups and requires each group is bound to the corresponding variable $\widetilde{PV} \in \rho(\lfloor x \rfloor)$, and (4) checks whether the pattern and the value list are of the same length or puts l in the error component and analyses the continuation

(TOut) $\rho, \kappa, \psi \models \langle E_1, \ldots, E_k \rangle.P$
iff $\wedge_{i=1}^{k}\ \rho \models E_i : \vartheta_i \wedge$
$\forall \widetilde{PV}_1, \ldots, \widetilde{PV}_k : \wedge_{i=1}^{k} \widetilde{PV}_i \in \vartheta_i \Rightarrow$
$(\langle fl(\biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_k)) \rangle \in \kappa \ \wedge\ \rho, \kappa, \psi \models P)$

(TInp) $\rho, \kappa, \psi \models (E_1, \ldots, E_j; x_{j+1}, \ldots, x_t)^l.P$
iff $\wedge_{i=1}^{j}\ \rho \models E_i : \vartheta_i \wedge$
$\forall \widetilde{PV}_1, \ldots, \widetilde{PV}_j : \wedge_{i=1}^{j} \widetilde{PV}_i \in \vartheta_i \Rightarrow$
$\forall \langle PV_1, \ldots, PV_k \rangle \in \kappa\ (k \geq len + t - j) :$
$(PV_1, \ldots, PV_{len}) = \biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_j) \Rightarrow$
$\forall (\widetilde{PV}'_{j+1}, \ldots, \widetilde{PV}'_t) \in \prod_{t-j}(PV_{k-len}, \ldots, PV_k) \Rightarrow$
$(\wedge_{i=j+1}^{t}\ \widetilde{PV}'_i \in \rho(\lfloor x_i \rfloor) \wedge (k > len + t - j \Rightarrow \lfloor l \rfloor \in \psi) \wedge$
$\rho, \kappa, \psi \models P)$
where $len = Len(\biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_j))$

(TDec) $\rho, \kappa, \psi \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}^l_{E_0}\ \mathsf{in}\ P$
iff $\rho \models E : \vartheta\ \wedge\ \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i \wedge$
$\forall \widetilde{PV}_1, \ldots, \widetilde{PV}_j : \wedge_{i=1}^{j} \widetilde{PV}_i \in \vartheta_i \Rightarrow$
$\forall \{PV_1, \ldots, PV_k\}_{\widetilde{PV}_0} \in \vartheta\ (k \geq len + t - j) \wedge$
$\widetilde{PV}_0 \in \vartheta_0 \wedge (PV_1, \ldots, PV_{len}) = \biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_j) \Rightarrow$
$\forall (\widetilde{PV}'_{j+1}, \ldots, \widetilde{PV}'_t) \in \prod_{t-j}(PV_{k-len}, \ldots, PV_k) \Rightarrow$
$(\wedge_{i=j+1}^{t}\ \widetilde{PV}'_i \in \rho(\lfloor x_i \rfloor) \wedge (k > len + t - j \Rightarrow \lfloor l \rfloor \in \psi) \wedge$
$\rho, \kappa, \psi \models P)$
where $len = Len(\biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_j))$

(TADec) $\rho, \kappa, \psi \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j; x_{j+1}, \ldots, x_k|\!\}^l_{E_0}\ \mathsf{in}\ P$
iff $\rho \models E : \vartheta\ \wedge\ \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i \wedge$
$\forall \{\!|PV_1, \ldots, PV_k|\!\}_{\widetilde{PV}_0} \in \vartheta : \forall \widetilde{PV}'_0 \in \vartheta_0 :$
$\forall (m^+, m^-) : \{\widetilde{PV}_0, \widetilde{PV}'_0\} = \{(\lfloor m^+ \rfloor), (\lfloor m^- \rfloor)\} \wedge$
$\forall \widetilde{PV}_1, \ldots, \widetilde{PV}_j : \wedge_{i=1}^{j} \widetilde{PV}_i \in \vartheta_i \Rightarrow$
$(PV_1, \ldots, PV_{len}) = \biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_j) \Rightarrow$
$\forall (\widetilde{PV}'_{j+1}, \ldots, \widetilde{PV}'_t) \in \prod_{t-j}(PV_{k-len}, \ldots, PV_k) \Rightarrow$
$(\wedge_{i=j+1}^{t}\ \widetilde{PV}'_i \in \rho(\lfloor x_i \rfloor) \wedge (k > len + t - j \Rightarrow \lfloor l \rfloor \in \psi) \wedge$
$\rho, \kappa, \psi \models P)$
where $len = Len(\biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_j))$

Table 9.3: Complex Type Flaw Analysis of LySA Processes: $\rho, \kappa, \psi \models P$

process.

The rules (TDec) and (TADec) do a similar job as (TInp): analyse E to get all the possible encrypted values, perform pattern matching and variable binding, and finally check the possibility of multi-to-one binding.

9.4 Correctness of the Complex Type Flaw Analysis

Lemma 9.6 (Substitution in expression) *$\rho \models E : \vartheta$ and $(\lfloor PV_1 \rfloor, \ldots, \lfloor PV_k \rfloor) \in \rho(\lfloor x \rfloor)$ imply $\rho \models E[PV_1, \ldots, PV_k/x] : \vartheta$*

Proof. The proof proceeds by structural induction over expressions by regarding each of the rules in the analysis.

Case (Name). Assume that $E = n$ and $\rho \models n : \vartheta$. For arbitrary choices of x and $PV_1, \ldots, PV_k$ it holds that $n[PV_1, \ldots, PV_k/x] = n$ so it is immediate that also $\rho \models n[PV_1, \ldots, PV_k/x] : \vartheta$.

Case (Public Key), (Private Key) are similar.

Case (Variable). Assume that $E = x'$ and $\rho \models x' : \vartheta$, i.e. that $\rho(\lfloor x' \rfloor) \subseteq \vartheta$. Then there are two cases. Either $x' \neq x$ in which case $x'[PV_1, \ldots, PV_k/x] = x'$ so clearly $\rho \models x'[PV_1, \ldots, PV_k/x] : \vartheta$. Alternatively, $x' = x$ in which case $x'[PV_1, \ldots, PV_k/x] = PV_1, \ldots, PV_k$. Furthermore assume that $(\lfloor PV_1 \rfloor, \ldots, \lfloor PV_k \rfloor) \in \rho(\lfloor x \rfloor)$ and because $\rho(\lfloor x' \rfloor) \subseteq \vartheta$, it holds that $\rho \models (PV_1, \ldots, PV_k) : \vartheta$ in which case $\rho \models x'[PV_1, \ldots, PV_k/x] : \vartheta$ by the analysis.

Case (Symmetric Encryption). Assume that $E = \{E_1, \ldots, E_k\}_{E_0}$, i.e. $\rho \models \{E_1, \ldots, E_k\}_{E_0} : \vartheta$. The result holds by applying the induction hypothesis on each individual E_i.

Case (Asymmetric Encryption) is similar.

Lemma 9.7 (Substitution in processes) *$\rho, \kappa \models P : \psi$ and $(\lfloor PV_1 \rfloor, \ldots, \lfloor PV_k \rfloor) \in \rho(\lfloor x \rfloor)$ imply $\rho, \kappa \models P[\lfloor PV_1 \rfloor, \ldots, \lfloor PV_k \rfloor/x] : \psi$*

Proof. The proof is done by straightforward induction applying the induction hypothesis on any sub-process and lemma 9.6 on any sub-terms.

Lemma 9.8 (Evaluation of values) *The analysis* $\rho \models PV : \vartheta$ *holds if and only if* $(\lfloor PV \rfloor) \in \vartheta$.

PROOF. The proof is by induction in the structure of values. Remembering that values, PV, are expressions without variables, the proof is straightforward.

Theorem 9.9 (Correctness of Complex Type Flaw Analysis) *If* $P \rightarrow Q$ *and* $\rho, \kappa, \psi \models P$ *then also* $\rho, \kappa, \psi \models Q$. *Furthermore, if* $\psi = \emptyset$ *then* $P \rightarrow_{RM} Q$

PROOF. By induction on the inference of $P \rightarrow Q$.

In case (Com) we assume that

$$\rho, \kappa \models \langle PV_1, \ldots, PV_k \rangle.P \mid (PV'_1, \ldots, PV'_j; x_{j+1}, \ldots, x_t)^l.Q$$

which amounts to:

$$\begin{array}{ll}
(a) & \wedge_{i=1}^{k} \rho \models PV_i : \vartheta_i \\
(b) & \forall \widetilde{PV}_1, \ldots, \widetilde{PV}_k : \wedge_{i=1}^{k} \widetilde{PV}_i \in \vartheta_i \Rightarrow \langle fl(\biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_k)) \rangle \in \kappa \\
(c) & \rho, \kappa, \psi \models P \\
(d) & \wedge_{i=1}^{j} \rho \models PV'_i : \vartheta'_i \\
(e) & \forall \widetilde{PV}_1, \ldots, \widetilde{PV}_j : \wedge_{i=1}^{j} \widetilde{PV}_i \in \vartheta_i \Rightarrow \\
 & \quad \forall \langle PV_1, \ldots, PV_k \rangle \in \kappa : (PV_1, \ldots, PV_{len}) = \biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_j) \Rightarrow \\
 & \quad\quad \forall (\widetilde{PV}'_{j+1}, \ldots, \widetilde{PV}'_t) \in \prod_{t-j} (PV_{k-len}, \ldots, PV_k) \Rightarrow \\
 & \quad\quad\quad (\wedge_{i=j+1}^{t} \widetilde{PV}'_i \in \rho(\lfloor x_i \rfloor)) \wedge \rho, \kappa, \psi \models P \wedge \\
 & \quad\quad\quad\quad (k > len + t - j) \Rightarrow \lfloor l \rfloor \in \psi) \\
 & \quad \text{where } len = Len(\biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_j))
\end{array}$$

Moreover we assume that $\wedge_{i=1}^{j} PV_i = PV'_i$ and $k \geq t$ because $\langle PV_1, \ldots, PV_k \rangle.P \mid (PV'_1, \ldots, PV'_j; x_{j+1}, \ldots, x_t).Q \rightarrow P \mid Q[\widetilde{PV}_{j+1}/x_{j+1}, \ldots, \widetilde{PV}_k/x_k]$ where $(\widetilde{PV}_{j+1}, \ldots, \widetilde{PV}_t) \in \prod_{t-j}(PV_{j+1}, \ldots, PV_k)$, and we have to prove that $\rho, \kappa, \psi \models P \mid Q[fl(\widetilde{PV}_{j+1})/x_{j+1}, \ldots, fl(\widetilde{PV}_k)/x_k]$.

From (a) we have $\wedge_{i=1}^{k}(PV_i) \in \vartheta_i$ because of Lemma 9.8 since $\wedge_{i=1}^{k} \mathsf{fv}(PV_i) = \emptyset$ and then (b) gives $\langle PV_1, \ldots, PV_k \rangle \in \kappa$ since $\biguplus((PV_1), \ldots, (PV_k)) = (PV_1, \ldots, PV_k)$. From (d) and the assumption $\wedge_{i=1}^{j} PV_i = PV'_i$ we get $\wedge_{i=1}^{j}(PV_i) \in \vartheta'_i$. Apparently, $\biguplus((PV_1), \ldots, (PV_j)) = (PV_1, \ldots, PV_j)$ and $len = j$. Now (e) gives $\forall (\widetilde{PV}'_{j+1}, \ldots, \widetilde{PV}'_t) \in \prod_{t-j}(PV_{k-j}, \ldots, PV_k) \Rightarrow \wedge_{i=j+1}^{t} \widetilde{PV}'_i \in \rho(\lfloor x_i \rfloor)$ and $\rho, \kappa, \psi \models Q$. The substitution result (Lemma 9.7) then gives $\rho, \kappa, \psi \models Q[fl(\widetilde{PV}'_{j+1})/x_{j+1}, \ldots, fl(\widetilde{PV}'_t)/x_t]$ and together with (c) this gives the required result.

For the second part of the result we observe that $(k > len + t - j) \Rightarrow l \in \psi$ follows from (e) and since $\psi = \emptyset$ and $len = j$ it must the case that $t = k$. Thus the condition of the rule (Com) are fulfilled for $\rightarrow_{RM}$.

In case (Dec) we assume

$$\rho, \kappa, \psi \models \mathsf{decrypt}\ \{PV_1, \ldots, PV_k\}_{E_0}\ \mathsf{as}\ \{PV_1', \ldots, PV_j'; x_{j+1}, \ldots, x_t\}^l_{E_0'}\ \mathsf{in}\ P$$

which amounts to:

$$\begin{array}{ll}
(f) & \rho \models \{PV_1, \ldots, PV_k\}_{E_0} : \vartheta \\
(g) & \rho \models E_0' : \vartheta_0' \wedge \wedge_{i=1}^{j} \rho \models PV_i' : \vartheta_i' \\
(h) & \forall \widetilde{PV}_0, \ldots, \widetilde{PV}_j : \wedge_{i=1}^{j} \widetilde{PV}_i \in \vartheta_i' \Rightarrow \\
 & \quad \forall \{PV_1, \ldots, PV_k\}_{\widetilde{PV}_0'} \in \vartheta \wedge \widetilde{PV}_0 \in \vartheta_0 \wedge \\
 & \quad (PV_1, \ldots, PV_{len}) = \biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_j) \Rightarrow \\
 & \quad\quad \forall(\widetilde{PV}'_{j+1}, \ldots, \widetilde{PV}'_t) \in \textstyle\prod_{t-j}(PV_{k-len}, \ldots, PV_k)) \Rightarrow \\
 & \quad\quad\quad (\wedge_{i=j+1}^{t}\ \widetilde{PV}'_i \in \rho(\lfloor x_i \rfloor)\ \wedge \rho, \kappa, \psi \models P \wedge \\
 & \quad\quad\quad (k > len + t - j) \Rightarrow \lfloor l \rfloor \in \psi) \\
 & \quad \text{where } len = Len(\biguplus(\widetilde{PV}_1, \ldots, \widetilde{PV}_j))
\end{array}$$

Moreover we assume that $E_0 = E_0', \wedge_{i=1}^{j} PV_i = PV_i'$ and $k \geq t$ because $\mathsf{decrypt}\ \{PV_1, \ldots, PV_k\}_{E_0}\ \mathsf{as}\ \{PV_1', \ldots, PV_j'; x_{j+1}, \ldots, x_t\}^l_{E_0'}\ \mathsf{in}\ P \rightarrow P[\widetilde{PV}_{j+1}/x_{j+1}, \ldots, \widetilde{PV}_k/x_k]$ where $(\widetilde{PV}_{j+1}, \ldots, \widetilde{PV}_t) \in \prod_{t-j}(PV_{j+1}, \ldots, PV_t)$, and we have to prove $\rho, \kappa, \psi \models P[\widetilde{PV}_{j+1}/x_{j+1}, \ldots, \widetilde{PV}_k/x_k]$.

From (f) we have $E_0) \in \vartheta_i$ and $PV_i \in \vartheta_i$ since $\mathsf{fv}(E_0) = \emptyset$ and $\wedge_{i=1}^{k} \mathsf{fv}(PV_i) = \emptyset$, and then (g) gives $\{PV_1, \ldots, PV_k\}_{E_0} \in \vartheta$ since $\biguplus((PV_1), \ldots, (PV_k)) = PV_1, \ldots, PV_k$. From (h) and the assumption $E_0 = E_0'$ and $\wedge_{i=1}^{j} PV_i = PV_i'$ we get $E_0 \in \vartheta_0'$ and $\wedge_{i=1}^{j}(PV_i) \in \vartheta_i'$. Apparently, $\biguplus((PV_1), \ldots, (PV_j)) = (PV_1, \ldots, PV_j)$ and $len = j$. Now (e) gives $\forall(\widetilde{PV}'_{j+1}, \ldots, \widetilde{PV}'_t) \in \prod_{t-j}(PV_{k-j}, \ldots, PV_k) \Rightarrow \wedge_{i=j+1}^{t}\ \widetilde{PV}'_i \in \rho(\lfloor x_i \rfloor)$ and $\rho, \kappa, \psi \models Q$. The substitution result (Lemma 9.7) then gives $\rho, \kappa, \psi \models P[\widetilde{PV}'_{j+1}/x_{j+1}, \ldots, \widetilde{PV}'_t/x_t]$, which is the required result.

For the second part of the result we observe that $(k > len + t - j) \Rightarrow l \in \psi$ follows from (h) and since $\psi = \emptyset$ and $len = j$ it must the case that $t = k$. Thus the condition of the rule (Dec) are fulfilled for $\rightarrow_{RM}$.

In case (ADec) and (ASig) are similar.

In cases (New), (ANew), (Par) and (Rep) follow directly from the induc-

tion hypothesis.

The case (Congr) also uses the congruence result.

9.5 The Attacker

The syntax of LySA has now been extended for modelling complex type flaws. Consequently, the way of modelling the attacker has to be changed correspondingly.

In the complex type flaw analysis, each message sent to or received by principals is viewed as a sequence of bits. Protocol participants then parse the bit sequences into a number of fields as specified by the protocol. It is reasonable to assume that each protocol participants know the bit length of the expected message beforehand and hence refuse to accept the messages of a different length. Under this assumption, we say that the attack can only send out messages of which the bit length equals to one of the messages exchanged during the protocol execution. We claim that the attacker will not gain anything more by sending out a message of a different bit length. Similarly for encryptions, the attacker only generates encryptions of which the bit length equals to an encryption generated during the protocol execution. Formally, we define a function, $BLen$, to represent the bit length of names and encryptions, i.e.

$$BLen(PV): \quad \text{the bit length of the value } PV$$

Furthermore, we require that

- the bit lengths of all the sending or receiving messages are in $\mathcal{B}_\kappa$, and
- the bit lengths of all the symmetric and asymmetric encryptions are in $\mathcal{B}_{\mathsf{Enc}}$

Obviously, $\mathcal{B}_\kappa$ and $\mathcal{B}_{\mathsf{Enc}}$ are all finite and can be computed by inspecting the process P_{sys}.

Given the assumptions as above, the extended Dolev-Yao condition for the LySA calculus can be expressed as the conjunction of the five components in Table 9.4.

Theorem 9.10 (Correctness of Dolev-Yao Condition) *If* (ρ, κ, ψ) *satisfies* $\mathcal{F}^{DY}_{RM}$ *of type* $(\mathcal{N}_{\mathsf{f}}, \mathcal{B}_\kappa, \mathcal{B}_{\mathsf{Enc}})$ *then* $\rho, \kappa, \psi \models \overline{Q}$ *for all attackers* Q *of extended type* $(\{z_\bullet\}, \mathcal{N}_{\mathsf{f}} \cup \{n_\bullet\}, \mathcal{B}_\kappa, \mathcal{B}_{\mathsf{Enc}})$.

(1) $\wedge_{k\in\mathcal{B}_\kappa}\ \forall\,\langle PV_1,\ldots,PV_k\rangle\in\kappa:\ \wedge_{i=1}^{k}(PV_i)\in\rho(z_\bullet)$
the attacker may learn by eavesdropping

(2) $\wedge_{k\in\mathcal{B}_{\mathsf{Enc}}}\ \forall(\{PV_1,\ldots,PV_k\}_{\widetilde{PV_0}})\in\rho(z_\bullet):\widetilde{PV_0}\in\rho(z_\bullet)\Rightarrow \wedge_{i=1}^{k}(PV_i)\in\rho(z_\bullet)$

$\wedge_{k\in\mathcal{B}_{\mathsf{Enc}}}\ \forall(\{\!|PV_1,\ldots,PV_k|\!\}_{m^+})\in\rho(z_\bullet):(m^-)\in\rho(z_\bullet)\Rightarrow \wedge_{i=1}^{k}(PV_i)\in\rho(z_\bullet)$

$\wedge_{k\in\mathcal{B}_{\mathsf{Enc}}}\ \forall(\{\!|PV_1,\ldots,PV_k|\!\}_{m^-})\in\rho(z_\bullet):(m^+)\in\rho(z_\bullet)\Rightarrow \wedge_{i=1}^{k}(PV_i)\in\rho(z_\bullet)$

the attacker may learn by decrypting messages with keys already known

(3) $\forall\ \widetilde{PV_0},PV_1,\ldots,PV_k:\widetilde{PV_0}\in\rho(z_\bullet)\wedge\wedge_{i=1}^{k}(PV_i)\in\rho(z_\bullet)\wedge BLen(\{PV_1,\ldots,PV_k\}_{\widetilde{PV_0}})\in\mathcal{B}_{\mathsf{Enc}}\Rightarrow(\{PV_1,\ldots,PV_k\}_{\widetilde{PV_0}})\in\rho(z_\bullet)$

$\forall\ m^+,PV_1\ldots,PV_k:(m^+)\in\rho(z_\bullet)\wedge\wedge_{i=1}^{k}(PV_i)\in\rho(z_\bullet)\wedge BLen(\{\!|PV_1,\ldots,PV_k|\!\}_{m^+})\in\mathcal{B}_{\mathsf{Enc}}\Rightarrow(\{\!|PV_1,\ldots,PV_k|\!\}_{m^+})\in\rho(z_\bullet)$

$\forall\ m^-,PV_1\ldots,PV_k:(m^+)\in\rho(z_\bullet)\wedge\wedge_{i=1}^{k}(PV_i)\in\rho(z_\bullet)\wedge BLen(\{\!|PV_1,\ldots,PV_k|\!\}_{m^-})\in\mathcal{B}_{\mathsf{Enc}}\Rightarrow(\{\!|PV_1,\ldots,PV_k|\!\}_{m^-})\in\rho(z_\bullet)$
the attacker may construct new encryptions using the keys known

(4) $\forall PV_1,\ldots,PV_k:\wedge_{i=1}^{k}(PV_i)\in\rho(z_\bullet)\wedge BLen(PV_1)+\ldots+BLen(PV_k)\in\mathcal{B}_\kappa\Rightarrow\langle PV_1,\ldots,PV_k\rangle\in\kappa$
the attacker may actively forge new communications

(5) $\{(n_\bullet),(m_\bullet^{\pm})\}\cup\mathcal{N}_f\subseteq\rho(z_\bullet)$
the attacker initially has some knowledge

Table 9.4: The Attacker's Capabilities in Complex Type Flaw Analysis.

Proof. By structural induction on $\overline{Q}$.

In case of (Inp), i.e. $\overline{Q}$ is the process of the form

$$(\overline{E}_1,\ldots,\overline{E}_j;x_{j+1},\ldots,x_t)^{l_\bullet}.P$$

and here we need to find $\vartheta_1,\ldots,\vartheta_j$ and show

(a) $\wedge_{i=1}^{j}\rho \models \overline{E_i} : \vartheta_i$ and
for all $\widetilde{PV_1}, \ldots, \widetilde{PV_j}$ that $\wedge_{i=1}^{j}\widetilde{PV_i} \in \vartheta_i$, and
for all $\langle PV_1, \ldots, PV_k\rangle \in \kappa$ with $(PV_1, \ldots, PV_{len}) = \biguplus(\widetilde{PV_1}, \ldots, \widetilde{PV_j})$, and:
for all $\widetilde{PV'_{j+1}}, \ldots, \widetilde{PV'_t} \in \prod_{t-j}(PV_{k-len}, \ldots, PV_k)$ that
(b) $\wedge_{i=j+1}^{t}\widetilde{PV'_i} \in \rho(\lfloor x_i \rfloor)$
(c) $(k > len + t - j) \Rightarrow l_\bullet \in \psi$
(d) $\rho, \kappa, \psi \models \overline{P}$

We choose $\vartheta_i (1 \leq i \leq j)$ as the least set such that $\rho \models \overline{E_i} : \vartheta_i$ and prove that $\vartheta_i \subseteq \rho(z_\bullet)$; intuitively, if $\overline{E_i}$ has free variables $z_1, \ldots, z_m$ then ϑ_i consists of all values $\lfloor \overline{E_i}[PV_1/z_1, \ldots, PV_m/z_m] \rfloor$ where $(PV_j)(1 \leq j \leq m) \in \rho(z_\bullet)$. Next consider $\langle PV_1, \ldots, PV_k\rangle \in \kappa$ and assume that $(PV_1, \ldots, PV_{len}) = \biguplus(\widetilde{PV_1}, \ldots, \widetilde{PV_j})$ where $\wedge_{i=1}^{j}\widetilde{PV_i} \in \vartheta_i$ and let $\widetilde{PV'_{j+1}}, \ldots, \widetilde{PV'_t} \in \prod_{t-j}(PV_{k-len}, \ldots, PV_k)$, and this takes care of (a). Since $\wedge_{i=1}^{j}\vartheta_i \subseteq \rho(z_\bullet)$, as above, we have $\wedge_{i=1}^{j}\widetilde{PV_i} \in \rho(z_\bullet)$ and by $\mathcal{F}_{RM}^{DY}$ we get $\wedge_{i=j+1}^{t}\widetilde{PV'_i} \in \rho(\lfloor x_i \rfloor)$. Since $\lfloor x_i \rfloor = z_\bullet$ this takes care of (b) and (c); furthermore $\overline{P}$ has extended type $(\{z_\bullet\}, \mathcal{N}_f \cup \{n_\bullet\}, \mathcal{B}_\kappa, \mathcal{B}_{ENC})$ and the induction hypothesis then takes care of (d).

In case of (Dec), i.e. $\overline{Q}$ is the process of the form

$$\mathsf{decrypt}\ \overline{E}\ \mathsf{as}\ \{\overline{E_1}, \ldots, \overline{E_j}; x_{j+1}, \ldots, x_k\}^{l_\bullet}_{\overline{E_0}}\ \mathsf{in}\ \overline{P}$$

and we need to find ϑ and $\vartheta_0, \ldots, \vartheta_j$ and show

(a) $\rho \models \overline{E} : \vartheta \ \wedge\ \wedge_{i=0}^{j}\rho \models \overline{E_i} : \vartheta_i$
for all $\widetilde{PV_1}, \ldots, \widetilde{PV_j}$ that $\wedge_{i=1}^{j}\widetilde{PV_i} \in \vartheta_i$, and
for all $\{PV_1, \ldots, PV_k\}_{\widetilde{PV_0}} \in \vartheta$ with
$\widetilde{PV_0} \in \vartheta_0$ and $(PV_1, \ldots, PV_{len}) = \biguplus(\widetilde{PV_1}, \ldots, \widetilde{PV_j})$, and:
for all $\widetilde{PV'_{j+1}}, \ldots, \widetilde{PV'_t} \in \prod_{t-j}(PV_{k-len}, \ldots, PV_k)$ that
(b) $\wedge_{i=j+1}^{t}\widetilde{PV'_i} \in \rho(\lfloor x_i \rfloor)$
(c) $\neg\mathsf{RM}(t, k) \Rightarrow l_\bullet \in \psi$
(d) $\rho, \kappa, \psi \models \overline{P} : \psi$

We choose ϑ as the least set such that $\rho \models \overline{E} : \vartheta$ and prove that $\vartheta \subseteq \rho(z_\bullet)$; intuitively, if $\overline{E}$ has free variables $z_1, \ldots, z_m$ then ϑ consists of all values $\lfloor \overline{E}[PV_1/z_1, \ldots, PV_m/z_m] \rfloor$ where $(PV_i) \in \rho(z_\bullet)$. We perform a similar development for $\vartheta_0, \ldots, \vartheta_j$. Next consider $\{PV_1, \ldots, PV_k\}_{\widetilde{PV_0}} \in \vartheta$ and assume that $(PV_1, \ldots, PV_{len}) = \biguplus(\widetilde{PV_1}, \ldots, \widetilde{PV_j})$ where $\wedge_{i=1}^{j}\widetilde{PV_i} \in \vartheta_i$ and let $\widetilde{PV'_{j+1}}, \ldots, \widetilde{PV'_t} \in \prod_{t-j}(PV_{k-len}, \ldots, PV_k)$, and this takes care of (a).

Since $\vartheta_0 \subseteq \rho(z_\bullet)$, as above, we have $\widetilde{PV_0} \in \rho(z_\bullet)$ and by $\mathcal{F}^{DY}_{RM}$ we get $\wedge^t_{i=j+1}\widetilde{PV_i'} \in \rho(\lfloor x_i \rfloor)$. Since $\lfloor x_i \rfloor = z_\bullet$ this takes care of (b) and (c); furthermore $\overline{P}$ has extended type $(\{z_\bullet\}, \mathcal{N}_f \cup \{n_\bullet\}, \mathcal{B}_\kappa, \mathcal{B}_{ENC})$ and the induction hypothesis then takes care of (d).

The remaining cases are similar.

Example 9.6 *Consider the protocol from Example 9.1,*

$$\begin{array}{rcll} A & \rightarrow & : & \{A, N\}_K \\ & \rightarrow & B: & \{K'\}_K \end{array}$$

The protocol can be modelled as the following,

$$\begin{array}{ll} & (\nu\, N)\langle A, \{A, N\}_K\rangle.0 \\ \mid & (A; x_{enc})^{l_1}.\mathsf{decrypt}\ x_{enc}\ \mathsf{as}\ \{; x_k\}^{l_2}_K\ \mathsf{in}\ 0 \end{array}$$

where the identity of the sender (or the expected sender), A, is included in the output (or input) message.

Semantically, the two parallel processes can make a communication, where the message $\langle A, \{A, N\}_K\rangle$ is sent onto the network, which results in binding the value $(\{A, N\}_K)$ to x_{enc} and the further decryption of x_{enc} then binds (A, N) to the variable x_k. Finally the process evolves into $0 \mid 0$.

The process gives rise to binding the value $(\lfloor A \rfloor, \lfloor N \rfloor)$ to the variable x_k. The analysis components ρ, κ and ψ have the following entries:

$$\begin{array}{ll} \langle \lfloor A \rfloor, \{\lfloor A \rfloor, \lfloor N \rfloor\}_{\lfloor K \rfloor}\rangle \in \kappa & \psi = \{\lfloor l_2 \rfloor\} \\ (\{\lfloor A \rfloor, \lfloor N \rfloor\}_{\lfloor K \rfloor}) \in \rho(\lfloor x_{enc} \rfloor) & (\lfloor A \rfloor, \lfloor N \rfloor) \in \rho(\lfloor x_k \rfloor) \end{array}$$

Now $\psi = \{\lfloor l_2 \rfloor\}$ shows the complex type flaw attack is successfully detected.

9.6 Complex Type Flaw Analysis at the Meta Level

To show in detail how the complex type flaws analysis works, we use the Amended Needham-Schroeder protocol Protocol, which is subject to a complex

type flaw attack, as an example. The analysis results show that the attack can be successfully captured. The narration of the Amended Needham-Schroeder is the following (please see Chapter 2.4.1 for a detailed explanation),

$$
\begin{array}{lll}
1. & A \rightarrow B: & A \\
2. & B \rightarrow A: & \{A, N_B\}_{K_B} \\
3. & A \rightarrow S: & A, B, N_A, \{A, N_B\}_{K_B} \\
4. & S \rightarrow A: & \{N_A, B, K, \{K, N_B, A\}_{K_B}\}_{K_A} \\
5. & A \rightarrow B: & \{K, N_B, A\}_{K_B} \\
6. & B \rightarrow A: & \{N\}_K \\
7. & A \rightarrow B: & \{N-1\}_K
\end{array}
$$

Below is an encoding of the protocol in a scenario where all the principals A_i and B_j share the same key K. The indexing parallel is used to describe parallel sessions where principal A_i communicates to principal B_j. Note that indices are also attached to labels.

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{let}\ X \subseteq S_1\ \mathsf{in}\ \mathsf{let}\ Y \subseteq S_2\ \mathsf{in}\ (\nu\ K)} \\
 & |_{i\in X}|_{j\in Y} & !\langle A_i, B_j, A_i\rangle.(B_j, A_i; x1_{ij})^{l1_{ij}}. \\
 & & (\nu\ Na_{ij})\langle A_i, S, A_i, B_j, Na_{ij}, x1_{ij}\rangle. \\
 & & (S, A_i; x2_{ij})^{l2_{ij}}.\mathsf{decrypt}\ x2_{ij}\ \mathsf{as}\ \{Na_{ij}, B_j; xk_{ij}, x3_{ij}\}_K^{l3_{ij}}\ \mathsf{in} \\
 & & \langle A_i, B_j, x3_{ij}\rangle. \\
 & & (B_j, A_i; x4_{ij})^{l4_{ij}}.\mathsf{decrypt}\ x4_{ij}\ \mathsf{as}\ \{; xn_{ij}\}_{xk_{ij}}^{l5_{ij}}\ \mathsf{in} \\
 & & \langle A_i, B_j, \{xn_{ij}-1\}_{xk_{ij}}\rangle.0 \\
| & |_{i\in X}|_{j\in Y} & !(A_i, B_j; y1_{ij})^{l6_{ij}}. \\
 & & (\nu\ Nb_{ij})\ \ \langle B_j, A_i, \{y1_{ij}, Nb_{ij}\}_K\rangle. \\
 & & (A_i, B_j; y2_{ij})^{l7_{ij}}.\mathsf{decrypt}\ y2_{ij}\ \mathsf{as}\ \{Nb_{ij}, A_i; yk_{ij}\}_K^{l8_{ij}}\ \mathsf{in} \\
 & & (\nu\ N_{ij})\langle B_j, A_i, \{N_{ij}\}_{yk_{ij}}\rangle. \\
 & & (A_i, B_j; y3_{ij})^{l9_{ij}}.\mathsf{decrypt}\ y3_{ij}\ \mathsf{as}\ \{N_{ij};\}_{yk_{ij}}^{l10_{ij}}\ \mathsf{in}\ 0 \\
| & |_{i\in X}|_{j\in Y} & !(A_i, S, A_i, B_i; zna_{ij}, z1_{ij})^{l11_{ij}}.\mathsf{decrypt}\ z1_{ij}\ \mathsf{as}\ \{A_i; znb_{ij}\}_K^{l12_{ij}}\ \mathsf{in} \\
 & & (\nu\ K')\langle S, A_i, \{zna_{ij}, B_j, K'_{ij}, \{znb_{ij}, A_i, K'\}_K\}_K\rangle.0
\end{array}
$$

Taking $S_1 = \{1\}$ and $S_2 = \{1, 2\}$ the analysis holds whenever

$$\{l6_{11}, l6_{12}\} \subseteq \psi$$

Thus, the analysis reports possible complex type flaw attacks such that in the input $(A_i, B_j; y1_{ij})^{l6_{ij}}$, a multi-to-one binding may occur.

To see that there really is a violation one must find an execution that leads to the above binding violation. This can be attempted for any two sessions taking their indexes from $\mathbb{N}_1$ and $\mathbb{N}_2$. Consider the following message exchange:

	A	B	*Attacker*
1.1	$\langle A_1, B_1, A_1 \rangle$	$(A_1, B_1; y1_{11})$	
1.2	$(B_1, A_1; x1_{11})$	$\langle B_1, A_1, \{y1_{11}, Nb_{11}\}_K \rangle$	
1.3	$\langle A_1, S, A_1, B_1, Na_{11}, x1_{11} \rangle$		$(A_1, S, A_1, B_1; Z_{Na_{11}}, Z_{x1_{11}})$
2.1		$(A_1, B_1; y1_{11})$	$\langle A_1, B_1, Z_{Na_{11}}, B_1, Z_K \rangle$
2.2		$\langle B_1, A_1, \{y1_{11}, Nb_{11}\}_K \rangle$	$(B_1, A_1; Z_{y_{11}})$
1.4	$(S, A_1; x2_{11})$. decrypt $x2_{11}$ as $\{Na_{11}, B_1; xk_{11}, x3_{11}\}_K$		$\langle S, A_1, Z_{y_{11}} \rangle$
1.5	$\langle A_1, B_1, x3_{11} \rangle$		$(A_1, B_1; Z_{x3_{11}})$
1.6	$(B_1, A_1; x4_{11})$ decrypt $x4_{11}$ as $\{; xn_{11}\}_{xk_{11}}$		$\langle B_1, A_1, Z_N \rangle$
1.7	$\langle A_1, B_1, \{xn_{11} - 1\}_{xk_{11}} \rangle$		$(A_1, B_1; Z_{xm_{11}})$

In the step 2.1, the following pattern matching can successfully take place in the input construct,

$$(A_1, B_1; y1_{12}) \mid \langle A_1, B_1, Z_{Na_{11}}, B_1, Z_K \rangle$$

and let the variable $y1_{11}$ become bound to the value (Na_{11}, B_1, Z_K), where Z_K is a value generated by the attacker. This is a multi-to-one binding, which is recorded by $l6_{11} \in \psi$, and represents a complex type flaw attack. In step 2.2, the principal B_1 sends the value tuple $\langle B_1, A_1, \{Na_{11}, B_1, Z_K, Nb_{11}\}_K \rangle$ onto the network, which is intercepted and replayed by the attacker in step 1.4 to the principal A_1. After the successful decryption,

$$\mathsf{decrypt}\ \{Na_{11}, B_1, Z_K, Nb_{11}\}_K\ \mathsf{as}\ \{Na_{11}, B_1; xk_{11}, x3_{11}\}_K$$

A_1 then believes the value, Z_k, which becomes bound to the variable xk_{11}, is the new session key generated by the server S. In the next 2 steps, A_1 uses Z_k as the new session key to finish the challenge and response procedure with the attacker and completes the protocol. Some of the relevant entries of the analysis components ρ and κ are shown below,

$$\begin{aligned}
\rho:\ & (A_1) \in \rho(y1_{11}) \\
& (\{A_1, Nb_{11}\}_K) \in \rho(x1_{11}) \\
& (Na_{11}, B_1, n_\bullet) \in \rho(y1_{11}) \\
& (n_\bullet) \in \rho(xk_{11}) \\
& (Nb_{11}) \in \rho(x3_{11})
\end{aligned}$$

$$\begin{aligned}
\kappa:\ & \langle A_1, B_1, Na_{11}, B_1, n_\bullet, \rangle \in \kappa \\
& \langle B_1, A_1, \{Na_{11}, B_1, n_\bullet, Nb_{11}\}_K \rangle \in \kappa
\end{aligned}$$

Similarly, the attack may happen to the session between the principals A_1 and B_2, which accounts for the error $l6_{12} \in \psi$.

The protocol can be modified such that each principal with different keys for different roles, i.e. all the principals A_i share a master key Ka_i with the server and all the principals B_j share Kb_j with the server. For a version of the process that has been modified in this way, the analysis holds for $\psi = \emptyset$ and thereby it guarantees absence of complex type flaw attacks.

We have shown how the analysis can capture complex type flaw attacks by checking the possibilities of multi-to-one binding. However the analysis results are not precise enough, i.e. every multi-to-one binding is regarded as a violation, even when it may not represent an actual attack.

The extended notation of variable binding can be combined with origin and destination annotations to capture violation of authentication property caused by complex type flaw attacks. Rather than focusing on multi-to-one variable binding, one may add origin and destination annotations to the process in question and let the control flow analysis captures possible violation to the authentication annotations.

Example 9.7 *Consider the encoding of the amended Needham-Schroeder protocol with origin and destination annotations,*

$$
\begin{array}{ll}
\textit{let } X \subseteq S_1 \textit{ in let } Y \subseteq S_2 \textit{ in } (\nu\ K) & \\
\quad |_{i\in X}|_{j\in Y} & !\langle A_i, B_j, A_i\rangle.(B_j, A_i; x1_{ij})^{l1_{ij}}. \\
& (\nu\ Na_{ij})\langle A_i, S, A_i, B_j, Na_{ij}, x1_{ij}\rangle. \\
& (S, A_i; x2_{ij})^{l2_{ij}}. \\
& \textit{decrypt } x2_{ij} \textit{ as } \{Na_{ij}, B_j; xk_{ij}, x3_{ij}\}_K^{l3_{ij}}[\textit{at } a1 \textit{ orig } \{s2\}] \textit{ in} \\
& \langle A_i, B_j, x3_{ij}\rangle. \\
& (B_j, A_i; x4_{ij})^{l4_{ij}}.\textit{decrypt } x4_{ij} \textit{ as } \{; xn_{ij}\}_{xk_{ij}}^{l5_{ij}}[\textit{at } a2 \textit{ orig } \{b3\}] \textit{ in} \\
& \langle A_i, B_j, \{xn_{ij} - 1\}_{xk_{ij}}\rangle.0 \\
| \quad |_{i\in X}|_{j\in Y} & !(A_i, B_j; y1_{ij})^{l6_{ij}}. \\
& (\nu\ Nb_{ij})\ \ \langle B_j, A_i, \{y1_{ij}, Nb_{ij}\}_K[\textit{at } b1 \textit{ dest } \{s1\}]\rangle. \\
& (A_i, B_j; y2_{ij})^{l7_{ij}}. \\
& \textit{decrypt } y2_{ij} \textit{ as } \{Nb_{ij}, A_i; yk_{ij}\}_K^{l8_{ij}}[\textit{at } b2 \textit{ orig } s3] \textit{ in} \\
& (\nu\ N_{ij})\langle B_j, A_i, \{N_{ij}\}_{yk_{ij}}[\textit{at } b3 \textit{ dest } \{a2\}]\rangle. \\
& (A_i, B_j; y3_{ij})^{l9_{ij}}.\textit{decrypt } y3_{ij} \textit{ as } \{N_{ij};\}_{yk_{ij}}^{l10_{ij}}[\textit{at } b4 \textit{ orig } \{a3\}] \textit{ in } 0 \\
| \quad |_{i\in X}|_{j\in Y} & !(A_i, S, A_i, B_i; zna_{ij}, z1_{ij})^{l11_{ij}}. \\
& \textit{decrypt } z1_{ij} \textit{ as } \{A_i; znb_{ij}\}_K^{l12_{ij}}[\textit{at } s1 \textit{ orig } \{b1\}] \textit{ in} \\
& (\nu\ K') \\
& \langle S, A_i, \{zna_{ij}, B_j, K'_{ij}, \\
& \quad \{znb_{ij}, A_i, K'\}_K[\textit{at } s3 \textit{ dest } \{b2\}]\}_K[\textit{at } s2 \textit{ dest } \{a1\}]\rangle.0
\end{array}
$$

The analysis results have a non-empty error component, i.e.

$$(b1, a1) \in \psi$$

which amounts to the same complex type flaw attack as described before.

9.7 Summary

A complex type flaw attack happens when a concatenation of fields in a message is interpreted as a single field. However, most existing protocol analysers do not take this kind of attack into consideration. Little research has been done on this topic, among which is the work by Meadows [64] where a procedure is proposed to determine whether or not type confusions are possible for a given protocol. The approach involves a search for all potential misinterpretations of two given protocol messages that are of equal bit-string length. The procedure also evaluates the likelihood of the misinterpretations by means of probability. As a complementary work, Long [59, 58] developed an approach to prove whether or not a known complex type flaw attack may really happen on a given protocol.

The approach exploits the data structure of the Z [84] specification language by constructing all the protocol messages from a common atomic primitive, and therefore it gains control over which message fields may or may not be confused.

In this chapter, we extended the notation of variable binding in the process calculus LySA from one-to-one to multi-to-one binding, thus making easier modelling the scenario where a list of fields is confused with a single field. The semantics of extended LySA makes use of a reference monitor to capture the possible multi-to-one binding at run time. On the static property, the control flow analysis is extended to take multi-to-one binding into account. A protocol is ensured as free of complex type flaws when the control flow analysis confirms the zero possibility of multi-to-one binding.

Chapter 10

Conclusion

We studied a number of properties of security protocols, in the process algebra framework of the LySA calculus, on the common basis of the operational semantics using dynamic and static techniques.

In the first four chapters, we presented some background information including an introduction to the process calculus LySA, a succinct and powerful language for modelling security protocols running in various scenarios, and a control flow analysis for the LySA calculus, which statically predicts safe and computable approximations of the behaviours that the object processes may be assumed to perform during the executions.

In the next five chapters, we considered various security properties and attacks. Each of them is formally expressed in terms of annotations. We then extended the control flow analysis in a way that the annotations are checked for compliance. The control flow analysis guarantees that if a process passes the test, it will never violate the security requirements at run time.

In this thesis, on top of process calculus modelling, five different control flow analyses have been presented. However, the analyses share a great amount of similarities, as they are all themselves extensions of a fairly standard one, which over-approximates the run-time protocol behaviours. The main dissimilarity among the analyses is the way that each kind of annotations is handled in order

to verify security properties. This observation gives rise to a few options for further research.

10.1 Combination of Analyses

This thesis has presented a contribution in the area of automated analysis of various security properties, e.g. confidentiality, freshness, simple type flaw and complex type flaw, of communication protocols running in a malicious environment. On one hand, obviously, extending the repertoire with more interesting properties, e.g. integrity, availability and non-repudiation, that can be analysed is an ongoing line of future research. However, on the other hand, instead of putting a great amount of effort into building such a large property repertoire, where some elements of it might be of less interest to certain users, one may look into finding a way to integrate the existing analyses in order to yield more comprehensive results with, hopefully, less resource consumption.

The feasibility of this line of future work is based on the fact that all the first four analyses presented in this thesis share the same framework and the main differences are in the way of handling various annotations. As a first attempt along this line, we shall apply the combination of freshness analysis and simple type flaw analysis to the Andrew Secure RPC protocol.

$$
\begin{array}{lll}
1. & A \rightarrow B: & A, \{NA\}_K \\
2. & B \rightarrow A: & \{NA+1, NB\}_K \\
3. & A \rightarrow B: & \{NB+1\}_K \\
4. & B \rightarrow A: & \{K', NB'\}_K
\end{array}
$$

As shown in chapter 8.6, simple type flaw analysis detects that the protocol is subject to a type flaw attack, where the principal A accepts the message $\{N_A + 1, N_B\}_K$, from step 2, in step 4 and takes the value $N_A + 1$ as the new key K'.

Nevertheless, the simple type flaw attack is not the only attack that the Andrew Secure RPC protocol is subject to. Applying the freshness analysis to the protocol gives a hint that, the attacker may replay a message, say, $\{K'_{old}, NB'_{old}\}_K$, of another session in step 4, and fool the principal A to accept an old session key K'_{old} as a new one.

In this sense, the combination the freshness analysis and simple type flaw analysis finds two kinds of attacks that each single one is not capable of, and mean-

while, only requires to be run once. The combination of the two analyses could be obtained by generalising the syntax and turning on both reference monitors in the semantics.

10.1.1 Toward a More General Framework

In the previous section, we have explored the possibility of combining the freshness analysis and the simple type flaw analysis, and shown that the combination gives rise to a more comprehensive result than the ones yielded by each individual analysis. Along this line, one might find it interesting to create a more general framework, that not only characterises the existing analyses, but also allows one to freely specify annotations for dealing with other security properties that may be of interest without the need to modify the analysis or design a new analysis.

10.2 User-Friendly Interface

The analyses presented in this thesis are aiming at assisting protocol designers in the development of high quality, preferably flaw-free, protocols. In practice, however, users have to encode protocols in LySA and therefore at least basic knowledge about process calculi is required. Consequently, the analyses are often hard to use for non-experts, which limits their direct and practical impact. This is actually a weak point for not only LySA but a lot of other formal analysis tools. One of the ways to address this problem is to provide developers of security protocols with a high-level interface for those formal analysis tools, such as the work done in Casper [61], CAPSL [28], CVS [34], and AVISS [7]. As far as LySA is concerned, in [21], an interface for the LySA tool is defined in the style of the Unified Modelling Language (UML). The system works in the following way: 1) allow the security protocols to be modelling in a consistent way in UML, 2) take the UML model as input and convert it to a corresponding LySA process annotated with the security properties specified in the UML model, 3) analyse the LySA process and output the analysis result.

However, the UML interface requires professional knowledge in the field of UML and thus is inconvenient for non-expert users. Furthermore, as protocols are more often published in the protocol narrations manner, it might be plausible to allow users to describe protocols in a more compact way, e.g. "$A \rightarrow B : \ message$".

In fact, a project aiming at integrating the SENSORIA tool case into a service oriented architecture (SOA) has been carried out. As the LySa tool is a member of the tool case, one of the goals of the project is to develop a high-level interface for the LySa tool and encapsulate it as one of the services (units) that can be distributed over a network and can be combined together and reused to create other applications.

Bibliography

[1] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[2] M. Abadi. Security protocols and their properties. *Foundations of Secure Computation*, pages 39–60, 2000.

[3] M. Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Fourth ACM Conference on Computer and Communications Security*, pages 36–47, 1997.

[4] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *Technical report, SRC DIGITAL*, June 1994.

[5] R. M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. *Lecture Notes in Computer Science*, 1877:380, 2000.

[6] R. M. Amadio and S. Prasad. The game of the name in cryptographic tables. *Asian Computing Science Conference*, pages 15–26, 1999.

[7] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Vigano, and L. Vigneron. The aviss security protocol analysis tool. *Proceedings of the International Conference on Computer Aided Verification*, pages 349–353, 2002.

[8] J. C. M. Baeten and W. P. Weijland. *Process algebra*. Cambridge University Press, 1990.

[9] H. P. Barendregt. Introduction to lambda calculus. *Aspenæs Workshop on Implementation of Functional Languages, Göteborg*, 1988.

[10] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

[11] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Proceeding of Theoretical Computer Science*, 37(1):77–121, 1985.

[12] G. Berthelot and R.Terrat. Petri nets theory for the correctness of protocols. pages 2497–2505, 1982.

[13] G. Bochmann and J. Gecsel. A unified method for the specification and verification of protocols. *Information Processing*, pages 229–234, 1977.

[14] C. Bodei, Linda Brodo, P. Degano, and H. Gao. Detecting and preventing type flaws at static time. *Journal of Computer Security: Special Issue on Security of Ad Hoc and Sensor Networks*, 2008 (to appear).

[15] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson. Automatic valication of protocol narration. *Procceeding of Computer Security Foundations Workshop XVI*, pages 126–140, 2003.

[16] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390, 2005.

[17] C. Bodei, P. Degano, H. Gao, and L. Brodo. Detecting and preventing type flaws: a control flow analysis with tags. *In Proceeding of 5th International Workshop on Security Issues in Concurrency (SecCo'07), ENTCS*, 194:3–22, 2007.

[18] M. Boreale. Symbolic trace analysis of cryptographic protocols. *Lecture Notes in Computer Science*, 2076:667–682, 2001.

[19] C. Boyd. Security architectures using formal methods. *IEEE journal on Selected Areas in Communications*, 11(5):694–701, 1993.

[20] M. Buchholtz. *Automated Analysis of Security in Networking Systems.* PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2004.

[21] M. Buchholtz, C. Montangero, L. Perrone, and S. Semprini. For-lysa: Uml for authentication analysis. *Lecture Notes in Computer Science*, pages 93–106, 2005.

[22] M. Bugliesi, R. Focardi, and M. Maffei. Authenticity by tagging and typing. *Proceeding of 2nd ACM Workshop on Formal Methods in Security Engineering*, 2004.

[23] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Technical report 39*, February 1989.

[24] V. A. Busam and D. E. Englund. Optimization of expressions in fortran. *Communications of the ACM*, 12(12):666–674, 1969.

[25] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[26] I. Cervesato, N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. *Proceeding of IEEE Computer Security Foundations Workshop*, pages 55–69, 1999.

[27] L. Chen. An interleaving model for real time. *Proceeding of Symposium on Logical Foundation of Computer Science*, pages 81–92, 1992.

[28] G. Denker and J. Millen. The capsl integrated protocol environment. *In DARPA Information Survivability Conference*, pages 207–221, 2000.

[29] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8), August 1981.

[30] D. E. Denning. Cryptography and data security. 1982.

[31] M. Diaz. Modeling and analysis of communication and cooperation protocols using petri net based models. *Computer Networks*, pages 419–441, 1982.

[32] M. Diaz and P. Azema. Petri net based models for the specification and validation of protocols. *Lecture Notes in Computer Science*, pages 229–234, 1985.

[33] D. Dolev and A. C. Yao. On the scurity of public key protocols. *Proceedings of the IEEE 22nd Annual Symposium on Foundations of Computer Science*, pages 350–357, 1981.

[34] A. Durante, R. Focardi, and R. Gorrieri. CVS: A compiler for the analysis of cryptographic protocols. *Proceedings of The 12th Computer Security Foundations Workshop*, 1999.

[35] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. *Proceeding of 24th IEEE Symposium on Foundations of Computer Science*, pages 34–39, 1983.

[36] H. Gao, P. Degano, C. Bodei, and H. R. Nielson. Detecting replay attacks by freshness annotations. *In Workshop on Issues in the Theory of Security (WITS '07)*, 2007.

[37] H. Gao, P. Degano, C. Bodei, and H. R. Nielson. A formal analysis for capturing replay attacks in communication protocols. *In Proceeding of 12th Annual Asian Computing Science Conference: Focusing on Computer and Network Security (ASIAN'07), Lecture Notes in Computer Science*, 4846:150–165, 2007.

[38] H. Gao and H. R. Nielson. Analysis of lysa-calculus with explicit confidentiality annotations. *In Proceedings of 20th International Conference on Advanced Information Networking and Applications (AINA'06)*, 2:39–43, 2006.

[39] A. Giacalone, C. Jou, and S. A. Amolka. Algebraic reasoning for probabilistic concurrent systems. *Proceedings of IFIP TC2 Working Conference on Programming Concepts and Methods*, pages 443–458, 1990.

[40] R. J. Glabbeek, S. A. Amolka, B. Steffen, and C. M. N. Tofts. Reactive, generative and stratified models of probabilistic processes. *Proceedings of 5th Annual IEEE Symposium on Logic in Cimputer Science*, pages 130–141, 1990.

[41] D. Gollmann. *Computer Security.* Wiley, 1999.

[42] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. *Proceedings of the 5th International Working Conference on Dependable Computing for Critical Applications*, pages 44–55, 1995.

[43] A. D. Gordon. Typing correspondence assertions for communication protocols. *Proceeding of Mathematical Foundations of Programming Semantics*, 2001.

[44] A. D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. *Proceeding of Computer Security Foundations Symposium*, 2001.

[45] A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Proceeding of Computer Security Foundations Symposium*, 2002.

[46] R. R. Hansen. A prototype tool for javacard firewall analysis. *Nordic Workshop on Secure IT-Systems*, 2002.

[47] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. *Proceedings of IEEE Real-Time Systems Symposium*, pages 278–287, 1990.

[48] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *Proceedings of the 13th Computer Security Foundations Workshop*, 2000.

[49] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computations*, pages 221–239, 1995.

[50] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[51] C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science, Prentice Hall, 1985.

[52] 2006 IEEE Std 802.16e 2005. Standard for local and metropolitan area networks part 16: Air interface for fixed and mobile broadband wireless access systems amendment 2: Physical and medium access control layers for combined fixed and mobile operation in licensed bands and corrigendum 1. 2006.

[53] K. Jensen. Coloured petri nets. basic concepts, analysis methods and practical use. *Proceedings of Monographs in Theoretical Computer Science*, 1997.

[54] G. Juanole, B. Algayre, and J. Dufau. On communication protocol modeling and design. *Lecture Notes in Computer Science*, pages 267–287, 1988.

[55] S. Kent. Encryption-based protection for interactive user/computer communication. *Proceedings of the Fifth Data Communications Symposium*, September 1997.

[56] F. Levi and C. Bodei. Security analysis of mobile ambients. *Proceeding of the Workshop on Issues in the Theory of Security 2000*, pages 18–23, 2000.

[57] Y. Li, W. Yang, and J. Huang. Preventing type flaw attacks on security protocols with a simplified tagging scheme. *Journal of Information Science and Engineering*, pages 59–84, 2005.

[58] B. W. Long. Formal verification of type flaw attacks in security protocols. *Proceeding of the Tenth Asia-Pacific Software Engineering Conference Software Engineering Conference*, 2003.

[59] B. W. Long. Formal verification of a type flaw attack on a security protocol using object-z. *Procceeding of 4th International Conference of B and Z Users, ZB*, pages 319–333, 2005.

[60] I. Lopez. The use of galileo to represent and analyze telecommunications protocols. *Proceeding of 2nd Europe Workshop of Applied Theory Petri Nets*, pages 377–410, 1981.

[61] G. Lowe. Casper: A compiler for the analysis of security protocols. *Proceedings of the 10th Computer Security Foundations Workshop*, 1997.

[62] E. S. Lowry and C. W. Medlock. Object code optimization. *Communications of the ACM*, 12(1):13–22, 1969.

[63] H. C. Lundh and V. Cortier. Tree automata with one memory set constraints and cryptographic protocols. *Theoretical Computer Science*, pages 143–214, 2005.

[64] C. Meadows. Identifying potential type confusion in authenticated messages. *Proceedings of Workshop on Foundation of Computer Security*, pages 75–84, 2002.

[65] J. K. Millen. Term replacement algebra for the interrogator. 1997.

[66] R. Millner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100:1–40.

[67] R. Milner. *A Calculus of Communicating Systems*. Springer Verlag, 1982.

[68] R. Milner. *Communication and concurrency*. International Series in Computer Science, Prentice Hall, 1989.

[69] F. Moller and C. Tofts. A temporal calculus of communicating systems. *Proceedings of the First International Conference on Concurrency Theory*, pages 401–4152, 1990.

[70] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), December 1978.

[71] F. Nielson, H. R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer Verlag, 1999.

[72] H. R. Nielson and F. Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. *Proceeding of the 24th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 332–345, 1997.

[73] H. R. Nielson and F. Nielson. Flow logic for constraint based analysis. *Proceeding of the 7th International Conference on Compiler Construction*, pages 109–127, 1998.

[74] H. R. Nielson and F. Nielson. Flow logic: A multi-paradigmatic approach to static analysis. the essence of computation - complexity, analysis, transformation. *Lecture Notes in Computer Science*, pages 223–244, 2002.

[75] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, pages 358–366, 1953.

[76] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is np-complete. *Proceeding of the 14th IEEE Computer Security Foundations Workshop*, pages 174–190, 2001.

[77] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[78] T.K. Shridharbhai. Probability and statistics with reliability, queuing, and computer science applications. 1982.

[79] E. Snekkenes. Roles in cryptographic protocols. *Proceedings of the 1992 IEEE Computer Security Symposium on Research in Secruity and Privacy*, pages 105–119, May 1992.

[80] F. J. W. Symons. *Modelling and Analysis of Communication Protocols using Numerical Petri Nets*. PhD thesis, Telecommunication Systems Group, Department of Electrinic Engineering Science, University of Essex, 1978.

[81] P. Syverson. A taxonomy of replay attacks. *Proceeding of Computer Security Foundations Symposium*, 1994.

[82] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *The Journal of Symbolic Logic*, page 370, 1955.

[83] T. Y. C. Woo and S. S. Lam. Authentication for distributed systems. *William Stallings, Practical Cryptography for Data Internetworks*, 1992.

[84] J. B. Wordsworth. Software development with z - a practical approach to formal methods in software engineering. *International Computer Science Series*, 1992.

[85] W. Yi. Real-time behaviour of asynchronous agents. *Proceedings of the First International Conference on Concurrency Theory*, pages 502–520, 1990.